TU WIEN Informatics

Untersuchung skalierbarer Praktiken für autonome Teams in Softwaregroßprojekten:

Kategorien, Kriterien und Best Practices

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Ing. Martin Schliefellner, B.Sc.
Matrikelnummer 0828600

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig

Wien, 8. September 2025

Martin Schliefellner

Thomas Grechenig

Erklärung zur Verfassung der Arbeit

Ing. Martin Schliefellner, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 16.09.2025

Schliefellner Martin

Danksagung

An dieser Stelle möchte ich meiner Familie herzlich für ihre Unterstützung während meines Studiums danken. Ein besonderer Dank gilt meiner Frau Katharina für ihre Geduld, ihre Hilfsbereitschaft und ihre beständige Unterstützung in dieser intensiven Zeit.

Mein aufrichtiger Dank gilt auch DDI Dr. Raoul Vallon für seine engagierte Betreuung, seine Geduld sowie seinen kompetenten Rat. Ohne seine fachliche Begleitung und das konstruktive Feedback wäre diese Arbeit in dieser Form nicht möglich gewesen.

Ebenso danke ich Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig, DI Dr. Mario Bernhart und Mag. Dr. Brigitte Brem für die gewährte Flexibilität und die wertvolle Unterstützung, die maßgeblich zum erfolgreichen Abschluss dieser Arbeit beigetragen haben.

Ebenso danke ich allen Interviewpartnerinnen und -partnern sowie den Ansprechpartnerinnen und -partnern für ihre Offenheit und die wertvollen Einblicke, die maßgeblich zum Gelingen dieser Untersuchung beigetragen haben. Mein Dank richtet sich auch an die Mitarbeiterinnen und Mitarbeiter des Instituts sowie an die Mitglieder der Forschungsgruppe, deren hilfreiche Beiträge und konstruktives Feedback von großem Nutzen waren.

Kurzfassung

In Softwaregroßprojekten arbeiten mehrere Dutzend bis über hundert Personen verteilt auf zahlreiche Teams gemeinsam an einem einzigen Produkt. Dabei sind verschiedenste Rollen beteiligt – von Entwicklern, Testern und Architekten über Product Owner bis hin zu Projektleitern. Die Herausforderung besteht darin, eine übergreifende Koordination und gemeinsame Ausrichtung sicherzustellen, ohne die Autonomie und Eigenverantwortung der einzelnen Teams einzuschränken. Autonome Teams gelten als Erfolgsfaktor moderner agiler Softwareentwicklung – ihre effektive Einbindung in komplexe Großprojekte erfordert jedoch angepasste organisatorische Strukturen und praktikable Lösungsansätze.

Die Arbeit identifiziert bewährte Praktiken (Engl. „Best Practices“) für autonome Teams in industriellen Softwaregroßprojekten und überprüft deren Skalierbarkeit im Abgleich mit Erkenntnissen aus kleineren agilen Softwareprojekten, für die entsprechende Praktiken von Hoda et al. beschrieben wurden. Ziel ist es, sowohl Übereinstimmungen als auch kontextuelle Unterschiede herauszuarbeiten, um fundierte und praxistaugliche Empfehlungen für den alltäglichen Einsatz autonomer Teams in komplexen Projektszenarien zu formulieren.

Die Untersuchung basiert auf drei Fallstudien aus unterschiedlichen Anwendungsbereichen – einem Versicherungsträger, einem internationalen Autohauskonzern und einem Projekt im Gesundheitsbereich. Insgesamt werden 15 semi-strukturierte Interviews geführt und mittels thematischer Analyse nach Braun & Clarke ausgewertet. Im Rahmen der Analyse entstanden 13 zentrale Themenkategorien, denen auf Basis kodierter Interviewsegmente insgesamt 23 konkrete Best Practices zugeordnet wurden. Neun von 13 Praktiken von Hoda et al. zeigen eine inhaltliche Übereinstimmung mit den identifizierten Best Practices und wurden als anwendbar in Softwaregroßprojekten bestätigt. Die Skalierbarkeitsanalyse verdeutlicht, welche bewährten Praktiken aus kleinen agilen Teams auch in komplexen Softwaregroßprojekten wirksam einsetzbar sind, und liefert damit fundierte Ansatzpunkte für die Übertragbarkeit etablierter Arbeitsweisen auf großindustrielle Kontexte. Die Zuordnung der Praktiken zu den Kategorien erfolgt auf Basis definierter Kriterien, wodurch eine konsistente und nachvollziehbare Strukturierung gewährleistet wird.

Die identifizierten Praktiken können Organisationen dabei unterstützen, effektive Rahmenbedingungen für Selbstorganisation, Koordination und kontinuierliche Verbesserung in verteilten Entwicklungsstrukturen zu schaffen.

Keywords: *Autonome Teams, Skalierbare Best Practices, Softwaregroßprojekte, Selbstorganisation*

Abstract

In large software projects, many people – from dozens to over a hundred – work together across multiple teams on a single product. Various roles are involved, such as developers, testers, architects, product owners, and project managers. The main challenge is to ensure coordination and alignment between teams without limiting their autonomy and responsibility too much. Self-organizing teams are seen as a key success factor in modern agile software development. However, integrating them effectively into complex large-scale projects requires organizational structures and practical approaches.

This study identifies effective best practices for self-organizing teams in large industrial software projects and evaluates their scalability by comparing them with findings from smaller agile projects, as described by Hoda et al. The goal is to highlight both common ground and context-specific differences, in order to develop well-founded and practical recommendations for the daily use of self-organizing teams in complex project environments.

The study is based on three case studies from different application domains – a social insurance institution, an international automotive corporation, and a healthcare-related software project. A total of 15 semi-structured interviews were conducted and analyzed using thematic analysis according to Braun & Clarke. The analysis resulted in 13 central thematic categories, to which a total of 23 specific best practices were assigned based on coded interview segments. Nine of 13 practices described by Hoda et al. show a clear conceptual match with the identified best practices and were confirmed to be applicable in large-scale software projects. The scalability analysis illustrates which proven practices from smaller agile teams can also be effectively applied in complex large-scale software projects, providing concrete guidance for transferring established work practices to industrial-scale development contexts. The assignment of practices to categories was carried out based on defined criteria, ensuring a consistent and transparent structure.

The identified practices aim to help organizations create the right conditions for self-organization, collaboration, and continuous improvement in distributed development environments.

Keywords: *Autonomous Teams, Scalable Best Practices, Large-Scale Software Projects, Self-Organization*

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel führt in das Thema der Arbeit ein und beschreibt die Problemstellung sowie die Motivation hinter der Untersuchung. Anschließend werden die Zielsetzung und die zugrunde liegenden Forschungsfragen erläutert. Zudem wird das methodische Vorgehen vorgestellt, das zur Beantwortung der Forschungsfragen gewählt wurde. Abschließend wird der Aufbau der Arbeit beschrieben, um dem Leser einen Überblick über die folgenden Kapitel zu geben.

1.1 Problemstellung

Geringe Autonomie und Autarkie von Projektmitarbeitern führt in traditionellen Projektstrukturen zu einem erhöhten Personalbedarf in der Führungsebene. Der Einsatz von autonomen Teams stellt einen Lösungsansatz dar, striktere Projektstrukturen zu vermeiden und thematisch abgegrenzte Aufgabenbereiche an die jeweiligen Teams abzugeben.

Die Übergabe dieser Aufgabenbereiche stellt hohe Anforderungen an die Teammitglieder und gefordert sind insbesondere Soft Skills, Erfahrung, Flexibilität, Verantwortungsbewusstsein und selbstständiges Arbeiten. Um diesen Anforderungen gerecht zu werden und effizientes Arbeiten zu ermöglichen, muss das Management den Teams ausreichend Freiraum gewähren, damit sie ihre Arbeitsweise bestmöglich an ihre Bedürfnisse anpassen können [1] [2].

Je größer ein Projekt wird, desto schwieriger wird der Koordinations- und Planungsaufwand. Der Begriff Softwaregroßprojekt (Engl. „large-scale software project“) wird von Dikert et al. als Projekt mit mindestens 50 Personen und 6 Teams beschrieben [3]. Als Folge der steigenden Komplexität bei Softwaregroßprojekten, bei denen sehr viele Teams an einem Produkt arbeiten, ergibt sich eine Einschränkung in der der Autonomie der einzelnen Teams. Um Softwaregroßprojekte managen zu können, muss ein einheitlicher Rahmen geschaffen werden. Vorgaben an Qualität, projektspezifische Standards und Release-Koordination gehören zu diesen Vorgaben [4] [5].

Obwohl die Teams in ihrem jeweiligen Verantwortungsbereich selbstständig arbeiten, ist eine Abstimmung in Architekturfragen sowie in Bezug auf Schnittstellen zu anderen Teams erforderlich. Zudem müssen sie den Projektfortschritt regelmäßig berichten und bei Bedarf gezielt fach-

liche Unterstützung einholen. Dies erhöht die Komplexität der Koordination zwischen den autonomen Teams erheblich. Insbesondere bei verteilten Standorten ist es entscheidend, dass klare Zuständigkeiten bestehen und der richtige Ansprechpartner bekannt ist [6] [7] [8].

In den letzten Jahren wurden im agilen Bereich verschiedene Softwareentwicklungsprozesse und Skalierungsframeworks für Softwaregroßprojekte geschaffen. SAFe, LeSS, Scrum-at-Scale, DAD oder das Spotify Model gehören zu diesen Frameworks zur Organisation agiler Entwicklungsprozesse in großen Unternehmen [9] [10] [11] [12] [13]. Unter anderem definieren diese Frameworks Muster, Praktiken und Leitlinien für ein Unternehmen, welches ein Softwaregroßprojekt umsetzen möchte. Dennoch sollten diese Skalierungsframeworks nicht nach Buch und ohne weitere Anpassung (Engl. „out of the box“) verwendet werden. Eher dienen sie als Inspiration für das Unternehmen und sollen bei der Entscheidungsfindung helfen. Eine Adaptierung an die Projektgegebenheiten ist daher unerlässlich [14].

Umso mehr ist es notwendig die Arbeitsweise der autonomen Softwareentwicklungsteams im Umfeld von Softwaregroßprojekten zu analysieren und Best Practices zu finden. Umfangreiche Arbeiten und Forschungen in diesem Umfeld werden von Hoda et al. geführt. Seit 2008 hat sie mit ihren Kollegen über 80 Publikationen im Umfeld autonome Teams und agile Softwareentwicklung geschaffen. Wenige neue Arbeiten der letzten vier Jahre widmen sich dem Umfeld in Softwaregroßprojekten und diese zeigen den großen Forschungsbedarf in diesem Umfeld auf [15].

Aufbauend auf den existierenden Forschungsergebnissen zu autonomen Teams im Umfeld von Softwaregroßprojekten, untersucht diese Arbeit die Arbeitsweise dieser Teams um Best Practices zu identifizieren. Als Abgrenzung ist festzuhalten, dass die Analyse rein auf der Arbeitsweise autonomer Teams vorgenommen wird. Die Arbeitsweise in der Führungsebene oder die Anwendbarkeit bestehender Frameworks ist nicht Teil dieser Arbeit.

Untersucht werden Praktiken zum Umgang der Aufwandsschätzung und Zuteilung von Aufgaben und Fehlern, der Reaktion auf fachliche und technische Änderungen, der Kommunikation der Teammitglieder und Tools, Gewährung der Transparenz, Fortschrittsmessung, Möglichkeiten der Spezialisierung und Weiterbildung sowie der Wissensverteilung.

Die gefunden Best Practices sollen bestehende Forschungsergebnisse bestätigen und gegebenenfalls erweitern bzw. eine Skalierbarkeit (siehe Punkt 2.2.4) aufzeigen.

1.2 Motivation

Mit zunehmender Größe eines Softwareprojekts steigen die Anforderungen an Struktur, Abstimmung und Koordination. Während in kleineren Vorhaben oft einfache agile Strukturen genügen, erfordern große Projekte angepasste organisatorische Rahmenbedingungen, um Selbstorganisation und effektive Zusammenarbeit in mehreren Teams zu ermöglichen. Die Wahl des passenden Vorgehensmodells – ob agil, traditionell oder hybrid – prägt dabei entscheidend, wie Planung, Kommunikation und Umsetzung gestaltet werden. Beim Vorgehensmodell unterscheidet man zwischen agilen Modellen (z.B. Scrum oder Extreme Programming), traditionellen Modellen (z.B. Wasserfall) oder hybriden Modellen (z.B. Scrumfall) [1] [16] [17]. Agile Vorgehensmodelle benötigen gegenüber traditionellen Modellen, selbstorganisierende Teamstrukturen, um zu funktionieren, wie in den zwölf Hauptprinzipien des agilen Manifests von Fowler und Highsmith [18] beschrieben.

Die Untersuchung der Arbeitsweise autonomer Teams in Softwaregroßprojekten stellt die Hauptmotivation des Autors dar, da es hier noch wenig relevante Forschung im großindustriellen Umfeld gibt. Es soll anhand der Arbeitsweisen „autonomer Teams“ untersucht werden, ob sich gemeinsame Best Practices in den zu untersuchenden Fallbeispielen finden lassen. Der State of the Art (siehe Kapitel 3) zeigt auf, unter welchen Rahmenbedingungen autonome Teams in Softwaregroßprojekten wirksam eingesetzt werden können und welche Herausforderungen dabei typischerweise auftreten. Diese Ergebnisse betreffen zum einen personelle Voraussetzungen und zum anderen Artefakte und Ressourcen, die zur Verfügung gestellt werden müssen, damit diese Art von Organisation funktioniert. Personelle Voraussetzungen betreffen Soft Skills, Erfahrung, Flexibilität, das Übernehmen von Verantwortung und selbstständiges Arbeiten. Zusätzlich zu diesen Eigenschaften, müssen den Teilnehmern die Ressourcen in ausreichendem Umfang zur Verfügung gestellt werden damit diese ihre Ziele erreichen können. Die Arbeitsumgebung ist für eine gute Kommunikation genauso wichtig, wie für Mitarbeiter motivierende Belohnungen.

Alle diese angeführten Faktoren gestalten sich in ihrer Umsetzung schwieriger je größer das Projekt ist. Anhand großindustrieller Fallbeispiele möchte der Autor aktuelle Forschungsergebnisse der Best Practices für autonome Teams in Ihrer Anwendbarkeit für Softwaregroßprojekte untersuchen. Die Motivation gilt dem Erfahrungsgewinn dieser Gegenüberstellung zwischen dem aktuellen State of the Art in kleineren Projekten mit autonomen Teams und den Ergebnissen dieser Arbeit für Softwaregroßprojekte. Die gewonnenen Erkenntnisse sollen autonome Teams in großen Softwareprojekten bei ihrer Organisation unterstützen.

1.3 Zielsetzung

Das Ziel dieser Diplomarbeit ist der Vergleich bestehender Forschungsergebnisse mit den Ergebnissen einer durchzuführenden Fallstudie, um bisherige Forschungsergebnisse im Kontext von Softwaregroßprojekten zu validieren bzw. Verbesserungspotential aufzuzeigen. Angeleitet wird die Diplomarbeit durch folgende Forschungsfragen (Research Questions - RQ):

RQ1a. Welche Praktiken lassen sich in Softwaregroßprojekten zur Unterstützung autonomer Teams identifizieren?
RQ1b. Welche Praktiken für autonome Teams sind auch für große Softwareprojekte skalierbar?
RQ2. In welche Kategorien können die identifizierten Praktiken unterteilt werden?
RQ3. Auf Basis welcher Kriterien kann eine Kategorisierung stattfinden?

Die durchzuführenden Fallstudien werden dem aktuellen State of the Art gegenübergestellt. Die Analyse zeigt exemplarisch, welche Praktiken in den untersuchten Projekten eingesetzt wurden und im Rahmen der Untersuchung als praktikabel für große Softwareprojekte identifiziert wurden. Die Untersuchung orientiert sich dabei an den thematischen Schwerpunkten der Problemstellung und analysiert deren Bedeutung im praktischen Anwendungskontext. Unter Berücksichtigung des aktuellen Forschungsstandes werden darauf aufbauend Kriterien zur Einordnung der identifizierten Praktiken formuliert, thematische Kategorien entwickelt und Best Practices sowie deren Skalierbarkeit identifiziert und bewertet.

1.4 Methodik

In dieser Arbeit wird ein qualitativer, fallstudienbasierter Forschungsansatz gewählt, um die Arbeitsweise autonomer Teams in Softwaregroßprojekten zu untersuchen. Ziel ist es, zentrale Best Practices zu identifizieren, deren Skalierbarkeit zu analysieren und diese systematisch auf Basis von definierten Kriterien in Kategorien einzuordnen. Die Untersuchung erfolgt kontextsensitiv und praxisnah anhand realer Projektbeispiele.

Einen grundlegenden theoretischen Rahmen liefert Robert K. Yin, der als einer der bedeutendsten methodischen Vertreter der Fallstudienforschung gilt. Seine Methodik wurde bereits in den 1980er-Jahren eingeführt und liegt mittlerweile in der 6. Auflage unter dem Titel “Case Study Research and Applications: Design and Methods” [19] vor.

Yin liefert für die Fallstudie folgende Definition:

> *„A case study is an empirical method that investigates a contemporary phenomenon (the 'case') in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident."* ([19], S. 15)

Dieses Verständnis ist besonders relevant für diese Arbeit, da sich die Selbstorganisation von Teams nicht isoliert vom jeweiligen Projektumfeld betrachten lässt. Auf dieser theoretischen Grundlage basiert der methodische Leitfaden von Runeson & Höst [20], welche die zentrale Prinzipien von Yin auf den Bereich des Software Engineerings übertragen und in ein praxisorientiertes Vorgehensmodell für empirische Studien überführen. Ihre Vorgehensweise ist besonders geeignet für Studien, die reale Softwareprojekte untersuchen.

Für die methodische Nachvollziehbarkeit wird dieses Kapitel in zwei Unterpunkte gegliedert:

- Abschnitt 1.4.1 beschreibt den theoretischen und methodischen Rahmen der Arbeit. Dabei wird das Forschungsdesign gemäß den fünf Phasen der Fallstudienforschung nach Runeson & Höst [20] erläutert. Diese systematische Herangehensweise bildet die methodische Grundlage für die vorliegende empirische Untersuchung.
- Abschnitt 1.4.2 beschreibt im Anschluss das konkrete Vorgehen in der Durchführung der Untersuchung. Dabei werden die einzelnen Erhebungsschritte, Datenquellen und Analyseverfahren im Detail dargestellt, wie sie im Rahmen dieser Arbeit umgesetzt werden.

Limitationen der Fallstudienforschung sowie der erzielten Ergebnisse werden in Kapitel 6.6 systematisch dargelegt.

1.4.1 Methodischer Rahmen

Die empirische Grundlage dieser Arbeit bildet die Fallstudienforschung nach Runeson et al. [20]. Diese Methode ist besonders gut geeignet, um komplexe Fragestellungen im realen Projektkontext zu untersuchen. Runeson et al. beschreiben dafür einen klaren Ablauf mit fünf Schritten, die übernommen und auf den konkreten Forschungskontext angewendet werden (vgl. Tabelle 1):

Schritt	Beschreibung nach Runeson & Höst (2008)	Konkretisierung in dieser Arbeit
1. Design der Fallstudie (Engl. „case study design")	Festlegung der Ziele und Forschungsfragen, Auswahl der zu untersuchenden Fälle.	Definition der Forschungsfragen RQ1a, RQ1b, RQ2 und RQ3. Auswahl von drei Softwaregroßprojekten mit autonomen Teams in unterschiedlichen Domänen (Versicherung, Automobilindustrie, Gesundheitswesen.

2. Vorbereitung der Datenerhebung (Engl. „preparation for data collection“)	Planung der Datenerhebung, Erstellung eines Leitfadens, Berücksichtigung ethischer Aspekte.	Entwicklung eines semi-strukturierten Interviewleitfadens auf Basis von bestehenden Kategorien aus der Literatur (z. B. Hoda et al.) und aktuellen Forschungsergebnissen (State of the Art), um eine vergleichbare Datengrundlage zu schaffen. Auswahl verschiedener Teilnehmerrollen (z. B. Entwickler, Architekt, Teamleitung), Vorbereitung ergänzender Artefaktanalyse (z. B. Dokumente, Boards).
3. Durchführung der Datenerhebung (Engl. „collecting evidence“)	Sammlung der Daten aus verschiedenen Quellen (Interviews, Beobachtungen, Dokumente).	Durchführung und Aufzeichnung von 5 Interviews je Fallbeispiel mit Teammitgliedern in unterschiedlichen Rollen. Ergänzend: Analyse von Artefakten sowie einer Participant Observation in einem Projekt zum besseren Verständnis der Teamabläufe.
4. Datenanalyse (Engl. „analysis of collected data“)	Strukturierte qualitative Auswertung, ggf. ergänzt durch quantitative Elemente.	Themenbasierte Analyse nach Braun & Clarke, unterstützt durch Softwaretools. Aufbau eines Systems aus Codes und Themen, die später als Grundlage für die Best Practices und Kategorien dienen. Zusätzlich erfolgt die Entwicklung der Kriterien zur Themenkategorisierung.
5. Berichterstattung (Engl. „reporting“)	Präsentation der Ergebnisse und Reflexion der Validität.	Strukturierte Darstellung der Ergebnisse in Bezug auf die Forschungsfragen und Diskussion über Limitationen.

Tabelle 1: Prozess der Fallstudienforschung nach Runeson & Höst

1.4.2 Methodisches Vorgehen

Aufbauend auf dem methodischen Rahmen (vgl. Punkt 1.4.1) gliedert sich das konkrete Vorgehen dieser Arbeit in sechs klar definierte Phasen, die im Folgenden im Detail beschrieben werden:

I. Analyse vorhandener Literatur und Forschung

Vorhandene Literatur im Bereich autonomer Teams wird analysiert, um das theoretische Grundgerüst zu schaffen. Aktuelle Forschungsergebnisse werden dokumentiert, um diese für die Prüfung der Anwendbarkeit in den Fallstudien vorzubereiten. Der Fokus bei der Auswahl der Literatur gilt insbesondere den Herausforderungen, sowie den bekannten Best Practices, mit denen sich autonome Teams konfrontiert sehen. Die hierbei erzielten Analyseergebnisse gelten dann gleichermaßen als Referenz für die durchzuführende Fallstudie.

II. Participant Observation in einem Softwaregroßprojekt

Durch die aktive Teilnahme und Beobachtung in einer der Fallstudien entsteht ein besseres Verständnis für Abläufe, Zusammenarbeit, Koordination und Fachbegriffe. Diese Form der Datenerhebung – auch Participant Observation [21] genannt – liefert wichtige Einblicke, die bei der Erstellung der Interviewfragen und der späteren Analyse unterstützen.

III. Datensammlung aus drei Softwaregroßprojekten anhand von semi-strukturierten Interviews

Aufgrund der in Phase I und II erhaltenen Informationen werden semi-strukturierte Interviews mit Mitarbeitern in drei verschiedenen Softwaregroßprojekten geführt, um Informationen über deren Arbeitsweise zu erheben. Die Anzahl der Mitarbeiter in diesen Projekten befindet sich zwischen 80 und 200 Personen; die Anzahl der Mitarbeiter der einzelnen autonomen Teams variiert zwischen 5 und 15 Personen. Fallstudie 1 ist ein Projekt für einen großen Versicherungsträger in Österreich, Fallstudie 2 für einen großen Autohersteller und Fallstudie 3 widmet sich dem Gesundheitsbereich in Deutschland. Durch gezielte Fragen wird das Interview gelenkt, um vergleichbare Informationen zu erheben, die später in der Analyse (Phase V) gegenübergestellt werden. Das Interview erfolgt mündlich, direkt und wird mit den einzelnen Gruppenmitgliedern verschiedener Rollen der autonomen Teams geführt [22].

IV. Sammlung von Informationen aus weiteren Datenquellen

Um die Ergebnisse aus Phase III besser einordnen und absichern zu können, werden verschiedene Ressourcen als zusätzliche Datenquellen herangezogen – etwa Task Boards, Dokumente, Ticketsysteme und Wikis. Diese Artefakte dienen als ergänzende Informationsquelle und unterstützen den Interviewer dabei, die in den Interviews genannten Praktiken besser nachvollziehen zu können (Daten-Triangulation nach Yin [19]).

V. Thematische Analyse von Artefakten und Beobachtungen im Rahmen einer Multiple Case Study

In diesem Abschnitt werden die Datenquellen der Phasen II bis IV zueinander in Beziehung gesetzt, strukturiert und zusammengefasst. Die Analyse wird mit der thematischen Inhaltsanalyse nach V. Braun & V. Clarke [23] durchgeführt, die sich besonders gut für praxisnahe Forschungsfragen eignet, bei denen aus qualitativen Daten zentrale Themen identifiziert werden sollen. Die Methode ist nicht an eine bestimmte Theorie oder Erhebungsmethode gebunden und ermöglicht es, unterschiedliche Datenarten – etwa Interviews, Beobachtungen oder

textbasierte Artefakte – in einem gemeinsamen Analyseprozess zu kombinieren. Damit eignet sie sich ideal für kontextsensitive Untersuchungen in komplexen Projektumfeldern wie Softwaregroßprojekten mit autonomen Teams.

Die thematische Analyse bietet eine hohe Flexibilität und Anpassungsfähigkeit, ist jedoch gleichzeitig durch klar definierte Durchführungsschritte und analytische Entscheidungen strukturiert abgegrenzt. Sie besteht aus sechs Arbeitsschritten, zwischen denen – im Gegensatz zu einem linearen Vorgehen – iterativ und reflexiv gewechselt werden kann. Dabei entwickeln sich aus dem analysierten Datenmaterial schrittweise zentrale Themen, die zugleich als Kategorien für die spätere Strukturierung der Best Practices dienen.

Beobachtungen aus der aktiven Teilnahme (Participant Observation) sowie Inhalte aus projektbezogenen Artefakten fließen ergänzend in die Analyse ein und erweitern die Interviewaussagen um praxisnahe Einblicke in den realen Projektkontext. Diese enge Verbindung zwischen der Datenauswertung und der Kategoriebildung stellt einen zentralen methodischen Vorteil dar.

Die Datenbasis für den ersten Arbeitsschritt der thematischen Analyse ergibt sich aus den transkribierten, semi-strukturierten Interviews, deren Fragen anhand der sieben Kategorien von Hoda et al. vorstrukturiert sind [24]. Die sieben Kategorien der drei „Balancing Acts“ von Hoda et al. bilden eine gute strukturierte Basis für diese Forschungsarbeit. Die Kategorien sind übersichtlich gestaltet und decken ein breites Spektrum für die Untersuchung der Arbeitsweisen, wie Entscheidungsfindung, Aufgabenzuteilung oder Wissensverteilung autonomer Teams ab. Dadurch kann ein Vergleich und mögliche Skalierbarkeit der Best Practices besser herbeigeführt werden [25] [26].

Unterstützend zu den transkribierten Interviews werden Artefakte aus den Fallstudien einbezogen. Im zweiten Arbeitsschritt werden sogenannte Codes aus dem Datenmaterial erzeugt. Die Codes sind Schlagwörter, welche den Inhalt der Phrasen des Interviews zusammenfassend abbilden. Die erstellten Codes werden im dritten Arbeitsschritt zu Motiven bzw. Themen (Engl. „themes“) zusammengefasst welche anschließend bei Bedarf hierarchisiert und miteinander verknüpft werden.

Die Kodierung erfolgt induktiv und semantisch, das heißt, die Codes werden direkt aus dem Datenmaterial abgeleitet und orientieren sich an den explizit geäußerten Aussagen der Interviewten. Ziel ist es, relevante Themen und wiederkehrende Muster offen zu identifizieren. Die thematische Analyse folgt somit dem datengetriebenen Vorgehen nach Braun & Clarke, ohne vorgegebene theoretische Kategorien als Grundlage für die Kodierung zu verwenden.

Um die Codes den bestehenden Themen zuordnen zu können, kommt ein Kriterienkatalog zum Einsatz. Die Zuordnung erfolgte sowohl auf Basis der von Hoda et al. beschriebenen sieben Kategorien, die für diese Arbeit übernommen wurden, als auch anhand der inhaltlichen Relevanz und der im Rahmen der thematischen Analyse vorgenommenen Codierung der Interviewdaten. Die Kriterien bauen auf dem Modell von Hoda et al. auf und werden – sofern erforderlich – um empirisch abgeleitete Aspekte aus den Fallstudien ergänzt. Zusätzlich wird geprüft, wie häufig bestimmte Codes in den Interviews auftreten und in welchem Kontext sie genannt werden. Lässt sich ein Code anhand dieser Kriterien keiner bestehenden Kategorie eindeutig zuordnen, entsteht ein neues Thema.

Im Schritt vier werden die erstellten Themen nochmals mit dem Datenmaterial abgeglichen, um Widersprüche auszuschließen. Im Arbeitsschritt fünf findet eine Verfeinerung der Themen und deren Struktur statt um anschließend im sechsten und letzten Schritt das Ergebnis im Bezug zur Forschungsfrage auszuarbeiten.

Nicht relevante Informationen für die Forschungsfragen werden in diesem Abschnitt aussortiert. Die Codierung und Auswertung wird softwareunterstützt durchgeführt.

Die thematische Analyse erfolgt in zwei Schritten: Zunächst wird jedes Fallbeispiel im Rahmen einer Single Case Study einzeln betrachtet und kodiert. Die Auswertung reicht dabei bis zur Erstellung eines Code-Baumes mit den zugehörigen Themen. Im zweiten Schritt wird die Multiple Case Study durchgeführt, bei der die Ergebnisse aus allen drei Fallbeispielen zusammengeführt und Gemeinsamkeiten untersucht werden. Zur Sicherstellung der Vergleichbarkeit wird bereits in der Einzelanalyse auf ein konsistentes Benennungsschema der Kategorien geachtet. Dies erleichtert die spätere Zusammenführung der Ergebnisse und die Wiederverwendung der Kategorienamen in der Multiple Case Study. Zusätzlich werden bereits in der Phase der thematischen Analyse erste Kriterien zur Kategorisierung vorbereitet.

Für das finale Ergebnis werden ausschließlich jene Themen und Codes berücksichtigt, die in allen drei Fallbeispielen auftreten. Nur aus dieser gemeinsamen Schnittmenge werden konkrete Praktiken abgeleitet und inhaltlich als Best Practices ausgearbeitet. Diese bilden die Grundlage für die Kategorien sowie die Untersuchung der Skalierbarkeit. Themen oder Codes, die nur in zwei Fallbeispielen vorkommen, werden zwar dokumentiert, aber nicht weiter analysiert oder in Form von Best Practices formuliert. Sie fließen daher nicht in die Hauptkategorien oder die Skalierbarkeitsbewertung ein.

VI. Gegenüberstellung zu dem „State of the Art“ und Zusammenfassung der Ergebnisse

Die aus Phase V erarbeiteten Analyseergebnisse werden im Kapitel 6 systematisch beschrieben. Die Praktiken werden in 13 thematische Kategorien eingeordnet und mit den Forschungsergebnissen von Hoda et al. [24] [25] [26] verglichen. Dabei werden charakteristische Merkmale der Praktiken nach Hoda et al. herangezogen, um eine fundierte Zuordnung zu ermöglichen. Die Übereinstimmungen sowie Abweichungen werden tabellarisch dargestellt, um daraus skalierbare Best Practices für autonome Teams in Softwaregroßprojekten abzuleiten.

Im Folgenden werden die zentralen Forschungsfragen (RQ1–RQ3) jeweils den zugehörigen Analysezielen und methodischen Herangehensweisen zugeordnet:

1. Forschungsfrage RQ1a – Ziel: Best Practices

Im Rahmen einer qualitativen Mehrfallstudie werden konkrete Best Practices für autonome Teams in Softwaregroßprojekten identifiziert. Die Grundlage bildet eine empirische Datenerhebung mittels 15 semi-strukturierter Interviews, die nach dem Verfahren der thematischen Analyse nach Braun & Clarke ausgewertet werden. Aus den kodierten Segmenten der drei Fallbeispiele werden auf Basis wiederkehrender Muster sowie unter Berücksichtigung des inhaltlichen Kontexts praxiserprobte Vorgehensweisen abgeleitet, die anschließend als Best Practices formuliert und thematisch kategorisiert werden.

2. Forschungsfrage RQ1b – Ziel: Skalierbare Best Practices

Auf Basis von RQ1a erfolgt eine vertiefte Analyse der identifizierten Praktiken im Hinblick auf ihre Skalierbarkeit. Die abgeleiteten Best Practices werden dazu dem Modell von Hoda et al. gegenübergestellt. Ziel ist es zu prüfen, welche Praktiken inhaltliche Übereinstimmungen aufweisen und sich sowohl in kleinen als auch großen Softwareprojekten bewährt haben und als skalierbar gelten können.

3. Forschungsfrage RQ2 – Ziel: Kategorisierungen und Beziehungen zu den Best Practices

Anhand der thematischen Analyse nach Braun & Clarke werden Kategorien und Beziehungen erstellt. Die ermittelten Themen der Analyse fungieren zugleich als analytische Kategorien.

4. Forschungsfrage RQ3 – Ziel: Kriterien der Kategorisierung

Die Kriterien für die Kategorisierung wurden initial auf Basis der Kategorienbeschreibungen nach Hoda et al. entwickelt und im Verlauf der thematischen Inhaltsanalyse weiter verfeinert. Für neu identifizierte Kategorien wurden die Kriterien im Zuge der Themenbildung innerhalb der thematischen Analyse nach Braun & Clarke eigens ausgearbeitet.

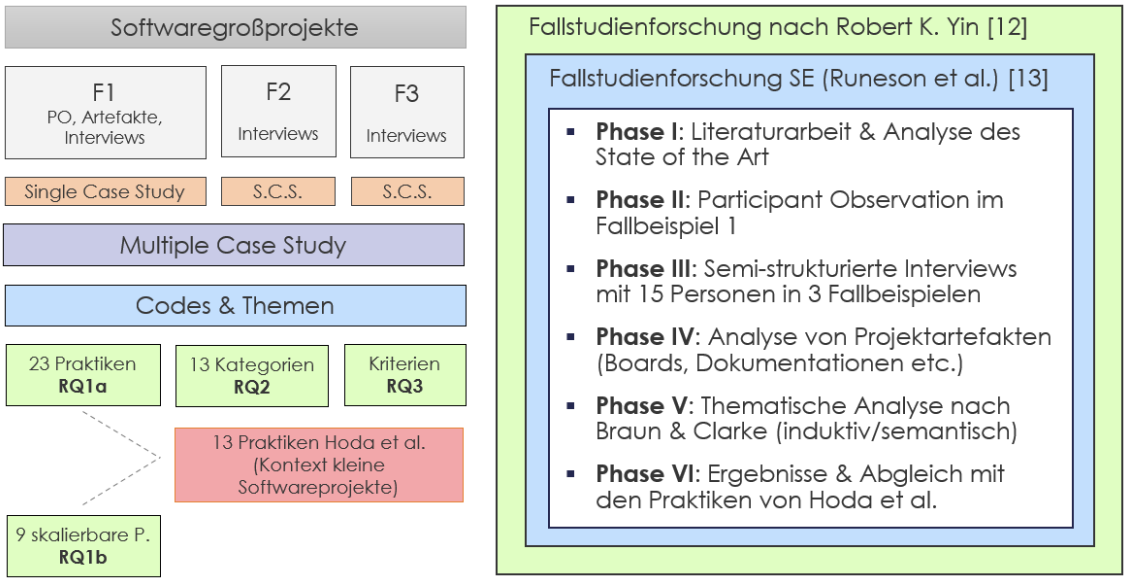


Abbildung 1: Illustration – Methodik

1.5 Aufbau der Arbeit

Kapitel 2 vermittelt die theoretischen Grundlagen, die für das Verständnis der Untersuchung notwendig sind. Es definiert zentrale Begriffe wie autonome Teams, industrielle Großsoftwareprojekte und skalierbare Best Practices. Zudem werden verschiedene Softwareentwicklungsmethoden vorgestellt, um den organisatorischen und methodischen Kontext der analysierten Projekte nachvollziehbar zu machen.

Kapitel 3 liefert einen Überblick über den aktuellen Stand der Forschung zu autonomen Teams in großen Softwareprojekten. Neben zentralen Herausforderungen und Rahmenbedingungen für die Skalierung agiler Zusammenarbeit werden auch aktuelle Skalierungsframeworks, organisationsbezogene Studien und empirische Forschungsergebnisse berücksichtigt. Einen besonderen Schwerpunkt bildet das Modell von Hoda et al., dessen Kategorien und Best Practices für selbstorganisierende agile Teams analysiert und im Kontext dieser Arbeit eingeordnet werden.

Kapitel 4 widmet sich der Untersuchung einzelner Fallbeispiele im Rahmen einer Single Case Study. Jedes Fallbeispiel wird unabhängig analysiert, um die spezifischen Arbeitsweisen autonomer Teams zu verstehen. Dazu gehören die Beschreibung der Projektorganisation, der Rollenverteilung, des Vorgehensmodells sowie die thematische Analyse der erhobenen Daten.

Kapitel 5 führt die Erkenntnisse aus den einzelnen Fallbeispielen in einer Multiple Case Study zusammen. Hierbei werden die Fallbeispiele vergleichend betrachtet, um übergreifende Muster

und Gemeinsamkeiten in den Arbeitsweisen autonomer Teams zu identifizieren. Die Ergebnisse helfen Best Practices abzuleiten und die gewonnenen Erkenntnisse zu verallgemeinern.

Kapitel 6 präsentiert die zusammengefassten Ergebnisse der Arbeit. Die aus den Fallstudien gewonnenen Erkenntnisse werden zunächst in Form von 23 Best Practices beschrieben (Forschungsfrage RQ1a, siehe Kapitel 6.1). Diese bilden die Grundlage für die Beantwortung der Forschungsfrage RQ1b: Welche Praktiken für autonome Teams sind auch für große Softwareprojekte skalierbar? – Diese Frage wird in Kapitel 6.2 behandelt. Anschließend werden die Praktiken thematisch in 13 Kategorien strukturiert (Forschungsfrage RQ2, siehe Kapitel 6.3). In Kapitel 6.4 werden schließlich Kriterien angeführt, anhand derer die Zuordnung der Best Practices zu den Kategorien erfolgt (RQ3). Kapitel 6.5 erweitert den Vergleich durch eine Gegenüberstellung mit aktuellen wissenschaftlichen Veröffentlichungen zum State of the Art, bevor in Kapitel 6.6 zentrale Limitationen der Ergebnisse reflektiert werden.

Kapitel 7 fasst die zentralen Ergebnisse zusammen. Abschließend wird ein Ausblick auf mögliche zukünftige Forschungsrichtungen gegeben.

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für diese Arbeit beschrieben. Kapitel 2.1 gibt eine kurze Einführung in Projektstrukturen. Im Kapitel 2.2 folgen Definitionen relevanter Begriffe für das Verständnis dieser Arbeit. Neben der allgemeinen Erklärung findet sich hier auch die Beschreibung wie diese im Bezug zu diese Forschungsarbeit zu verstehen sind.

2.1 Projektstrukturen

Das Ausmaß, in dem Teams autonom und autark arbeiten, hängt eng mit der gewählten Projektstruktur zusammen. In funktionalen Projektstrukturen – typischerweise in Form von Stabsprojektorganisationen – sind sowohl die Entscheidungsfreiheit (Autonomie) als auch die Unabhängigkeit von anderen Einheiten (Autarkie) der Teams stark eingeschränkt.

Autonomie bezeichnet die Fähigkeit eines Teams, innerhalb eines definierten Rahmens eigenständig Entscheidungen zu treffen. Autarkie beschreibt hingegen die strukturelle Unabhängigkeit – beispielsweise durch eigenen Ressourcenzugriff oder eigenverantwortliche Umsetzung. Beide Merkmale beeinflussen die Selbstorganisation, betreffen aber unterschiedliche Dimensionen.

In Matrixprojektorganisationen, etwa in Form von Leichtgewichts- oder Schwergewichtsorganisationen, sind Autonomie und Autarkie der Teams teils vorhanden, aber begrenzt. Während in der Leichtgewichtsorganisation die Projektleitung hauptsächlich koordinierende Funktionen erfüllt und die Linienorganisation dominiert, erhalten Projektteams in der Schwergewichtsorganisation mehr Entscheidungsspielraum und Verantwortung. Dennoch bleibt die Projektverantwortung meist klar hierarchisch geregelt, mit zentral vorgegebenen Rollen und Zuständigkeiten.

Reine Projektorganisationen ermöglichen dem Projektteam hingegen vollständige Verantwortung und weitgehende Unabhängigkeit von der Linienstruktur. In solchen Strukturen sind sowohl Autonomie als auch Autarkie stark ausgeprägt. Die Teams agieren eigenständig und können flexibel auf Veränderungen reagieren [27].

In Projektstrukturen mit geringer Autonomie und Autarkie entsteht häufig ein erhöhter Personalbedarf auf der Führungsebene, da viele Entscheidungen zentral getroffen werden müssen. Um

diese strukturellen Engpässe zu vermeiden, bietet sich der Einsatz autonomer Teams an. In solchen Strukturen werden themenspezifische Aufgaben gezielt an die jeweiligen Teams übergeben, was eine effizientere, dezentrale Arbeitsweise ermöglicht.

2.2 Definitionen relevanter Begriffe

In diesem Kapitel werden zentrale Begriffe erläutert, die für das Verständnis dieser Arbeit wichtig sind. Dazu zählen unter anderem autonome Teams, Softwareentwicklungsmethoden, Skalierbarkeit und Best Practices. Die Begriffe werden kurz eingeordnet und erklärt, um eine gemeinsame Ausgangsbasis zu schaffen. Ziel ist es, ein klares Bild davon zu vermitteln, was in dieser Arbeit jeweils unter den Begriffen verstanden wird.

2.2.1 Single case study vs. Multiple case study

Die Definition des Begriffs Fallstudie (Engl. „case study“) wird von John W. Cresswell wie folgt beschrieben:

> *„The case study method 'explores a real-life, contemporary bounded system (a case) or multiple bounded systems (cases) over time, through detailed, in-depth data collection involving multiple sources of information… and reports a case description and case themes'"* ([28], S.97)

Fallstudien können in zwei Typen unterteilt werden: Einzelfallstudien (Engl. „single case studies") oder Fallstudien mit mehreren Fällen (Engl. "multiple case studies"). Die Auswahl des Typs unterliegt dem Ziel des Forschungsgegenstandes und weiteren Kriterien wie zum Beispiel Zeit und Budget. Die Einzelfallstudie benötigt meist weniger Zeitaufwand und Budget. Zudem erlaubt die Einzelfallstudie dem Forscher eine höhere Qualität und Tiefe für die Materie. Möchte der Forscher ein bestimmtes Objekt oder Beziehungen zu diesem Objekt untersuchen, ist diese Fallstudie in der Regel die richtige Wahl.

Im Gegensatz dazu erlauben Multiple Case Studies eine Untersuchung verschiedener Einzelfälle und anschließend eine Analyse über alle diese Fälle. Somit können Gemeinsamkeiten und Differenzen eruiert und erfasst werden. Da bei dieser Methode mehrere Fallstudien untersucht werden und die Daten aus verschiedenen Quellen stammen, gewinnt die Analyse an Aussagekraft, da sie nicht auf ein einzelnes Objekt beschränkt ist [29].

2.2.2 Autonomie und Autarkie von Teams

In der englischsprachigen Literatur werden für autonome Teams häufig die Begriffe „self-directed teams“, „self-managed work teams“ oder „cross-functional teams“ verwendet. Sie bezeichnen Arbeitsgruppen, die einen Großteil ihrer Aufgaben eigenverantwortlich planen, durchführen und koordinieren – mit minimaler direkter Anleitung von außen.

Unter einem autonomen Team versteht man in diesem Zusammenhang eine Arbeitsgruppe von Personen, die für die Planung und Umsetzung technischer Aufgaben verantwortlich ist und die in einem Produkt oder Service für interne oder externe Abnehmer resultieren [30].

Kimball Fisher konkretisiert diesen Ansatz wie folgt:

> *„Self-directed team (noun): A group of employees who have day-to-day responsibility for managing themselves and the work they do with a minimum of direct supervision. Members of self-directed teams typically handle job assignments, plan and schedule work, make production- and/or service-related decisions, and take action on problems.”* ([30], S. 17)

In dieser Arbeit wird unter einem autonomen Team eine Arbeitsform verstanden, bei der Planung, Aufgabenverteilung, Umsetzung und Qualitätskontrolle weitgehend innerhalb des Teams erfolgen. Entscheidungen werden gemeinsam getroffen, die Arbeitsweise wird eigenständig organisiert und notwendige Anpassungen im Ablauf werden innerhalb des Teams vorgenommen – etwa bei der Einschätzung von Aufgaben, der Auswahl technischer Lösungen oder der Abstimmung von Prioritäten.

Während sich Autonomie auf die operative Entscheidungsfreiheit innerhalb vorgegebener Rahmenbedingungen bezieht, bezeichnet Autarkie eine weitergehende strukturelle Unabhängigkeit, beispielsweise durch eigene Ressourcenverantwortung oder die Möglichkeit, Entscheidungen weitgehend ohne externe Abhängigkeiten zu treffen.

In welchem Maß Autonomie und Autarkie in der Praxis realisiert werden können, ist insbesondere in Softwaregroßprojekten durch bestimmte Rahmenbedingungen begrenzt. Faktoren wie die Anzahl der beteiligten Personen, bestehende Schnittstellen, externe Abhängigkeiten oder zentrale Vorgaben beeinflussen den Handlungsspielraum der Teams maßgeblich.

2.2.3 Industrielles Großsoftwareprojekt

Ab wann ein Softwareprojekt als groß gilt hängt vom Standpunkt des Betrachters ab. Diverse Kriterien wie Anzahl der Teilnehmer, Dauer des Projektes, Budget, technische Komplexität, Unsicherheiten bei Zielen, Umgebung des Projektes, externe Einflüsse, Projektstruktur und wirtschaftliches Risiko sind nur einige Faktoren die Projekte charakterisieren. Die Einflussfaktoren der Komplexität auf ein Projekt werden nach dem „Task- and Orientation Model“ (TO) in zwei große Kategorien geteilt: Komplexität der Aufgaben (Engl. „task complexity factors”) und organisatorische Einflüsse (Engl. “organization complexity factors”).

Die Komplexität der Aufgaben fasst unter anderem die technischen Anforderungen hinsichtlich Know-how, Wechselwirkungen zwischen den Aufgaben, funktionale und nicht-funktionale Anforderungen an das System, Wechselwirkung mit der Umwelt (externe Einflüsse), Ressourcenverwaltung und Sicherstellung des Informationsflusses zwischen allen Stakeholdern zusammen.

Die Komplexität der Organisation beinhaltet die Anzahl der Teilnehmer mit ihren Führungsfertigkeiten, technischen Know-how und koordinatorischen Fähigkeiten sowie deren Erfahrung.

Je größer ein Projekt ist, desto schwieriger wird es, die oben angeführten Faktoren miteinander in Einklang zu bringen. Oft beinhaltet ein großes Softwareprojekt mehrere Subprojekte, die für sich wiederum einer eigenen Planung unterliegen. Fehlen gewisse Ressourcen können Teile des Projektes an externe Firmen ausgelagert werden oder es müssen externe Personen in die Entwicklung miteinbezogen werden.

Bei Softwaregroßprojekten muss der „versteckten Arbeitsaufwand“ (Engl. „hidden workload“) berücksichtigt werden. Dieser ergibt sich aufgrund der zunehmenden Komplexität und Abhängigkeit zwischen verschiedenen Subprojekten und Aufgaben, deren notwendigen Überarbeitung, Koordination und Wartezeiten bis zur Fertigstellung. Um diesen versteckten Arbeitsaufwand zu reduzieren, sind eine klare Kommunikation zwischen den Beteiligten, ausreichendes technisches und organisatorisches Know-how, eine präzise Dokumentation der Anforderungen sowie eine geeignete organisatorische Struktur von zentraler Bedeutung [31] [32].

In dieser Forschungsarbeit wird ein Projekt als Softwaregroßprojekt bezeichnet, wenn es folgende Merkmale aufweist, die deutlich über den Rahmen gewöhnlicher Softwareentwicklungsprojekte hinausgehen:

- eine hohe organisatorische Komplexität, beispielsweise durch zahlreiche Themenbereiche, Subprojekte oder Arbeitspakete

- eine Projektgröße mit über 80 beteiligten Personen und mehr als 5 autonomen Teams aus unterschiedlichen fachlichen und technischen Bereichen
- eine Laufzeit von mehr als vier Jahren, typischerweise mit lang angelegten Entwicklungs-, Wartungs- und Anpassungsphasen
- eine technisch anspruchsvolle Systemlandschaft, gekennzeichnet durch eine Vielzahl an eingesetzten Frameworks, Technologien und Schnittstellen zu externen Systemen
- eine umfangreiche Codebasis mit über einer Million Zeilen Quellcode
- sowie eine hohe wirtschaftliche Relevanz, bei der ein Scheitern des Projekts mit erheblichen finanziellen Folgen verbunden wäre – etwa durch Produktverzögerungen, Abbrüche oder Nachbesserungskosten.

Die genauen Charakteristika zu den Projekten der einzelnen Fallstudien dieser Arbeit werden im jeweiligen Abschnitt angeführt.

2.2.4 Skalierbare Best Practices

In der Informatik bezeichnet Skalierbarkeit die Fähigkeit eines Systems, bei zunehmender Last, Komplexität oder Ressourcennutzung leistungsfähig zu bleiben oder entsprechend erweitert werden zu können. Unterschieden wird dabei üblicherweise zwischen horizontaler Skalierbarkeit, also der Erweiterung durch zusätzliche Ressourcen wie Server oder Instanzen, und vertikaler Skalierbarkeit, bei der vorhandene Komponenten wie Prozessorleistung oder Speicher vergrößert werden [33].

In dieser Arbeit wird der Begriff Skalierbarkeit auf bewährte Arbeitspraktiken (Best Practices) übertragen. Eine Best Practice gilt hier dann als skalierbar, wenn sie sich nicht nur in kleineren, selbstorganisierten Teams bewährt hat, sondern auch in komplexeren Softwaregroßprojekten lt. Definition 2.2.3, mit längerer Laufzeit, größerer Teamanzahl und erhöhter technischer sowie organisatorischer Komplexität zur Anwendung kommt.

Die konzeptionelle Grundlage bildet die Grounded-Theory-Studie von Hoda et al., in der auf Basis von 58 Interviews in 23 Unternehmen typische Praktiken selbstorganisierter Teams in Softwareprojekten untersucht wurden [24]. Von den insgesamt analysierten Projekten in Hodas Studie hatten zwölf eine Laufzeit zwischen 1 und 12 Monaten; zwei weitere dauerten 36 bzw. 48 Monate. Die jeweiligen Teams bestanden aus 4 bis 15 Personen. Die in diesem Rahmen identifizierten Best Practices dienen in dieser Arbeit als Referenzmodell, um zu untersuchen, ob sich vergleichbare Praktiken auch in größeren und komplexeren Softwareprojektumgebungen wiederfinden.

Die Analyse der Fallstudien erfolgt dabei nicht im Rahmen einer Grounded-Theory-Methodik, sondern mittels thematischer Analyse nach Braun & Clarke, um zentrale Themen und wiederkehrende Praktiken herauszuarbeiten. Wenn in den untersuchten Softwaregroßprojekten Praktiken identifiziert werden, die mit denen aus der Studie von Hoda et al. übereinstimmen, gelten diese im Sinne dieser Arbeit als skalierbar.

2.2.5 Softwareentwicklungsmethoden

Im Folgenden werden Softwareentwicklungsmethoden beschrieben, welche für das Verständnis dieser Arbeit von Bedeutung sind. Sie werden in dieser Arbeit aufbauenden Literatur und Forschung erwähnt oder kommen in den Fallbeispielen zum Einsatz.

2.2.5.1 Scrum

Scrum gehört zu den iterativen und inkrementellen Vorgehensmodellen in der Softwareentwicklung. Dieser Prozess wurde unter der Annahme entwickelt, dass Softwareentwicklung zu komplex und unvorhersehbar ist, um vollständig im Voraus geplant werden zu können. Die vielen verschiedenen technischen und fachlichen Variablen in einem Softwareprojekt müssen während der gesamten Projektlaufzeit kontinuierlich beobachtet und bei Bedarf angepasst werden. Zu diesen Variablen zählen unter anderem eingesetzte Technologien und Werkzeuge, Anforderungen, Qualität, das Budget oder der Zeitplan. Die Abbildung 2 zeigt den Ablauf eines Scrum-Prozesses.

Im sogenannten Product Backlog befinden sich die priorisierten und aufwandsgeschätzten Anforderungen für die Funktionalität des Produktes, welche bereits für Releases eingeplant sind. Diese werden vom Product Owner verwaltet und gemeinsam mit den Teams in den Sprintplanning Meetings für den nächsten Sprint zugeteilt. Ein Sprint ist ein festgelegter Zeitraum, in welchem die zugeteilten Anforderungen entwickelt werden. Dafür wandern diese Anforderungen aus dem Product Backlog in den Sprint Backlog. Das Entwicklungsteam ist für die Umsetzung dieser Anforderungen zuständig.

Die Teams sind selbstorganisierend, verfügen über das nötige fachliche Know-how und übernehmen die Verantwortung für die vollständige Umsetzung ihrer Aufgaben. In jeder Iteration wird die Funktionalität des Produkts erweitert („Increment of functionality“).

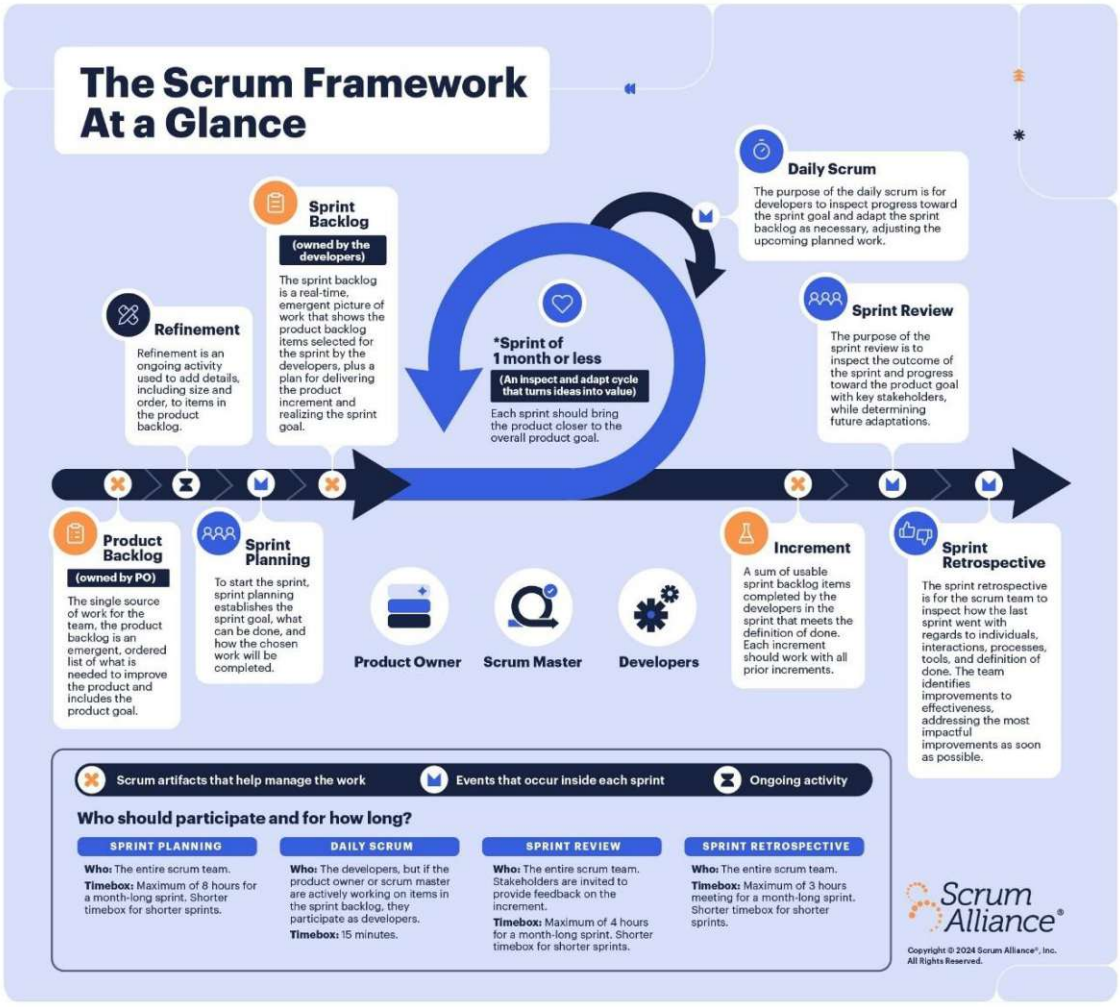


Abbildung 2: Scrum-Prozess [34]

Ein ca. 15-minütiges tägliches Meeting – das Daily Scrum – hilft bei der Überprüfung, Beobachtung und Adaptierung der umzusetzenden Aufgaben sowie Erkennung auftretender Probleme. Es trägt dazu bei, die Erfolgschancen der Teams bei der Zielerreichung zu verbessern. Eine weitere Rolle spielt der Scrum Master. Dieser trägt die Verantwortung für den Scrum-Prozess. Er ist für die Einhaltung, die Umsetzung und die Adaptierung des Scrum-Prozesses im Projekt zuständig sowie der Vermittlung der zugehörigen Regeln und des Ablaufs. Am Ende des Sprints wird die neue Funktionalität demonstriert. Weiteres wird ein Sprint Review durchgeführt. In diesem wird festgehalten was erledigt wurde und was nicht erreicht wurde, welche Probleme aufgetreten sind und wie diese gelöst werden konnten. Diese Informationen sind wichtig für die Planung der nächsten Sprints.

Nach dem Sprint Review und vor der Planung des nächsten Sprints erfolgt die Sprint Retrospektive. In der Retrospektive reflektiert das Team die Zusammenarbeit im vergangenen Sprint sowie den Einsatz von Werkzeugen und Technologien. Ziel ist es, konkrete Verbesserungen für den

nächsten Sprint abzuleiten. Dadurch können zukünftige Sprints gezielt optimiert und bereits identifizierte Probleme proaktiv adressiert werden.

Durch die Möglichkeit Anforderungen im Product Backlog zu verwerfen, neue Anforderungen hinzuzufügen sowie Prioritäten umzugestalten ist dieses Modell besonders flexibel [34] [35] [36].

2.2.5.2 Extreme Programming (XP)

Wie Beck et al. im Buch „Extreme Programming Explained“ beschreiben, ist Extreme Programming (XP) ein leichtgewichtiges Vorgehensmodell für kleine bis mittelgroße Teams, das insbesondere bei sich rasch ändernden Anforderungen zum Einsatz kommt:

> *„XP is a lightweight methodology for small-to-medium sized teams developing software in the face of vague or rapidly changing requirements.”* [37]

Die Agile Alliance hat eine etwas andere Definition für Extreme Programming:

> *„Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.”* [38]

Extreme Programming kommt bei Softwareprojekten zum Einsatz, bei denen kleinere autonome Teams eng zusammenarbeiten und bei denen auf ständig ändernde Anforderungen schnell reagiert werden muss.

Die wichtigsten Werte (XP-Values) sind Einfachheit (Engl. „simplicity“), Kommunikation (Engl. „communication“), Rückmeldung (Engl. „feedback“) und Mut (Engl. „courage“) [39] [40]. Beck et al. und die Agile Alliance fügen diesen Values noch den Punkt Respekt (Engl. „respect“) hinzu.

Neben diesen fünf zentralen Werten gibt es ein Set von 12 Kernpraktiken mit denen XP-Teams arbeiten. Diese Praktiken werden in Abbildung 3 dargestellt. Der innerste Kreis umfasst zentrale Programmierpraktiken wie Design Improvement, Test-driven Development, Simple Design und Pair Programming. Der mittlere Kreis dient der Kommunikation und Koordination und Qualitätssteigerung der Software. Der äußerste Kreis beschreibt Praktiken für die Planung der Softwarezyklen und Abstimmung mit dem Kunden [39].

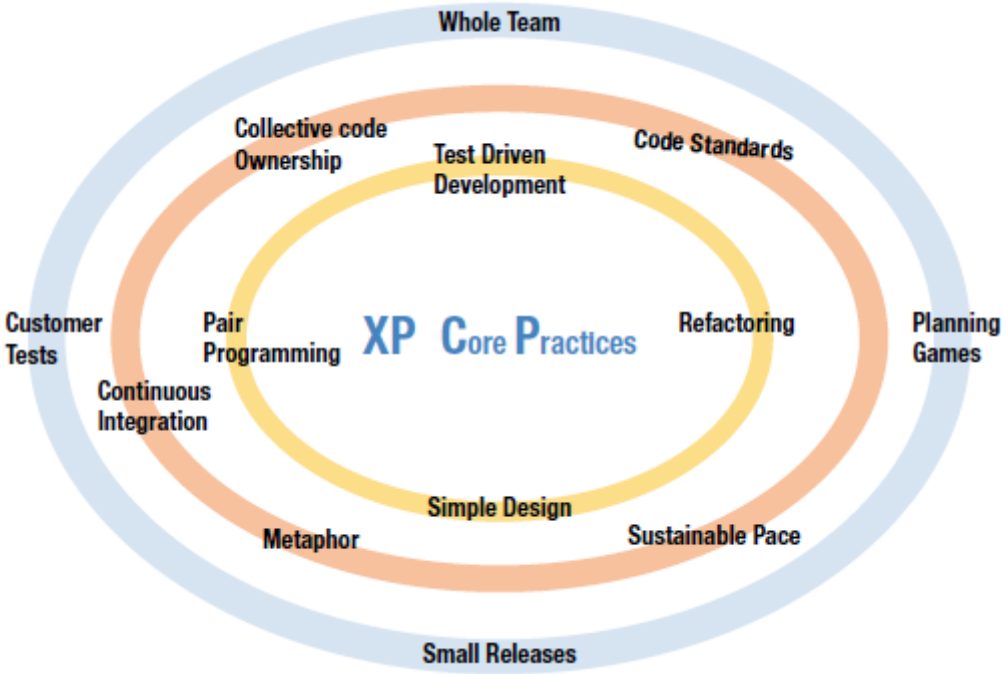


Abbildung 3: Die 12 Kernpraktiken von XP [41]

XP gliedert sich insgesamt in sechs Phasen: „Exploration", „Planning", „Iteration to release phase", „Production", „Maintenance" und die „Death Phase".

In der ersten Phase „Exploration" werden die Anforderungen an Design und Architektur, Tools und an die Software gestellt. Zusätzlich werden die Anforderungen in sogenannten „User-Stories" erfasst.

Die Phase „Planning", auch „Planning Game" genannt, beinhaltet die zwei Phasen „Iteration-Planning" und „Release-Planning". Im Release-Planning wird evaluiert welche Anforderungen in welchem Release umgesetzt werden. Die Auswahl erfolgt anhand von Prioritäten, Kapazität, Risikofaktoren und Zeitschätzungen. Im Iteration-Planning werden anschließend die wichtigsten Anforderungen zu einzelnen Iterationen für die Umsetzung zugeteilt.

In der Phase „Iteration to release phase" erfolgt die Umsetzung, das Testen und die Integration der Anforderungen. Die „Production"-Phase liefert die entwickelten Anforderungen aus. Die letzte Phase, „Death-Phase", genannt kann auf zwei Arten erreicht werden: Alle Funktionalitäten wurden erfolgreich umgesetzt oder die Umsetzung ist gescheitert.

XP ist sehr flexibel und kann für das spezifische Projekt den Bedürfnissen angepasst und erweitert werden [41] [42].

2.2.5.3 Wasserfall-Modell

Das Wasserfall-Modell wurde bereits im Jahre 1970 von Winston W. Royce als Softwareentwicklungsprozess beschrieben. Es ist ein sequenzielles Modell mit den fünf aufeinanderfolgenden Projektphasen Analyse, Design, Implementierung, Test und Instandhaltung welche in Abbildung 4 (vgl. [1]) dargestellt sind.

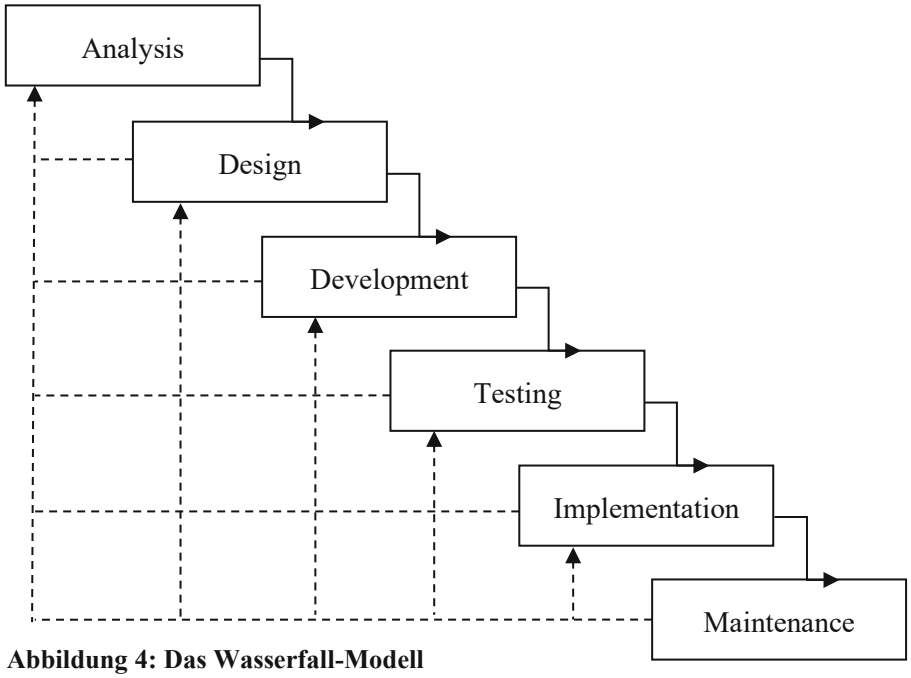


Abbildung 4: Das Wasserfall-Modell

Der Name des Wasserfallmodells ergibt sich aus seiner typischen Darstellung, bei der die Projektphasen als aufeinanderfolgende Stufen angeordnet sind. Es handelt sich um ein sequenzielles Vorgehensmodell, bei dem eine Phase formal erst dann beginnt, wenn die vorhergehende vollständig abgeschlossen ist. In vielen Darstellungen entsteht dadurch der Eindruck, dass Rücksprünge zu früheren Phasen nicht vorgesehen sind.

Tatsächlich weist Winston W. Royce in seinem ursprünglichen Artikel von 1970 [43] ausdrücklich darauf hin, dass Rückkopplungen zwischen den Phasen notwendig sind – insbesondere um Risiken im Entwicklungsprozess frühzeitig zu erkennen und zu vermeiden. Er bezeichnet das strikt sequenzielle Vorgehen als riskant, da viele Probleme erst spät im Prozess sichtbar werden. Um Risiken zu minimieren, empfiehlt er, zentrale Entwicklungsphasen wie Design und Imple-

mentierung mindestens zweimal zu durchlaufen. Iterative Elemente sind somit bereits im ursprünglichen Konzept enthalten, auch wenn spätere Adaptionen diese Rückkopplungen deutlicher hervorheben.

Die erste Projektphase ist die Analysephase. Hier werden sowohl funktionale als auch nicht-funktionale Anforderungen sowie die Spezifikationen der Software definiert. Anschließend folgt die Designphase, in der das Datenbankschema, die Softwarearchitektur, Algorithmen, Schnittstellen sowie das grafische Design ausgearbeitet werden. In der Implementierungsphase erfolgt die Programmierung der Software. Nach der Entwicklung beginnt die Testphase, in der überprüft wird, ob die ursprünglich definierten Anforderungen korrekt umgesetzt wurden. Ist der Test erfolgreich abgeschlossen, erfolgt die Auslieferung der Software. Anschließend werden in der Wartungsphase auftretende Fehler behoben und die Qualität und Performance weiter verbessert [1].

2.2.5.4 Kanban

Kanban wurde von Microsoft und Corbis von Scrum aus weiterentwickelt. Kanban ist agil und sehr flexibel. Das sogenannte Kanban-Board visualisiert die Arbeitsschritte der Aufgaben. Das Board beinhaltet mehrere Spalten, deren Bezeichnungen beliebig an die Gegebenheiten des Teams oder Projekts angepasst werden können. Ganz links befindet sich der Backlog, mit allen geplanten Aufgaben (Tasks). Die nächste Spalte wird als WiP („Work in Progress“, deutsch „in Arbeit“) bezeichnet. Weitere Spalten können zum Beispiel „Review“, „Test“, „Qualitätssicherung“ oder „Dokumentation“ sein. Am Ende befindet sich meist eine Spalte mit dem Status „Abgeschlossen“.

Der Workflow geht von links nach rechts. Die umzusetzenden Tasks werden z.B. anhand von „Post-Its“ aus dem Backlog „gepulled“ und in die Spalte „in Arbeit“ geschoben. Nach der Umsetzung geht das Ticket weiter bis zur Spalte „Abgeschlossen“. Eine Priorisierung der Aufgaben kann horizontal durch sogenannte „Swimlanes“ visualisiert werden. Je weiter oben sich ein Ticket am Kanban-Board befindet, desto höher ist es priorisiert [44]. Eine vereinfachte Darstellung eines Kanban-Boards wird in folgender Abbildung 5 dargestellt.

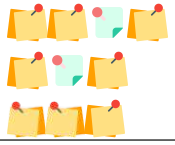
Backlog	In Arbeit	Review	Test	Abgeschlossen
				

Abbildung 5: Kanban-Board

Kanban definiert folgende 6 Praktiken:

1. Visualisierung der Arbeit: Die Arbeitsabläufe werden visualisiert
2. Limitierung der WiP-Tickets pro Team oder Team-Mitglied: Jede Spalte darf nur eine maximale Anzahl an Tickets enthalten
3. Management-Flow: Blockaden und Engpässe sollen erkannt und beseitigt werden. Die sogenannte „Cycle-Time“ (Zeit pro Ticket) soll reduziert werden.
4. Regulierung/Definition von Prozessregeln
5. Feedback-Loops: Feedback innerhalb des Teams und vom Kunden werden besprochen und eingearbeitet
6. Kaizen/Kontinuierliche Verbesserung [45] [46]

3 State of the Art

Die Popularität autonomer Teams stieg durch den Einsatz agiler Softwareentwicklungsmethoden wie zum Beispiel Scrum oder Extreme Programming (XP). Bereits Fowler und Highsmith führten 2001 in ihrem Manifest für agile Entwicklung selbstständig arbeitende Teams als eines der zwölf Hauptprinzipien an [18] [36].

Die drei fundamentalen Bedingungen für die Selbstorganisation – „Autonomy“, „Cross-fertilization“ und „Self-transcendence“ – wurden von Nonaka und Takeuchi bereits 1986 identifiziert [47].

Hoda, Noble und Marshall griffen diese Bedingungen in den Jahren 2009–2012 auf und entwickelten auf dieser Basis ihr Konzept der Balancing Acts. Das daraus abgeleitete Modell beschreibt drei Ebenen: konkrete teaminterne Praktiken (Engl. „Balancing Acts“), kontextspezifische Voraussetzungen (Engl. „Specific Conditions“) und organisationale Rahmenbedingungen (Engl. „General Conditions“) – und verbindet damit konkrete Praktiken mit den zugrunde liegenden Voraussetzungen für Selbstorganisation in agilen Teams [24] [25] [26] [48].

Die Balancing Acts von Hoda et al. definieren 13 Best Practices auf unterster Ebene (Engl. „low-level practices“) für autonome Teams, welche auf sieben Kategorien und drei übergeordnete Spannungsfelder (Balancing Acts) aufgeteilt sind. Diese 13 Praktiken bilden eine gute Basis für das Arbeiten autonomer Teams, wurden jedoch nicht im Umfeld von Softwaregroßprojekten untersucht.

Aktuelle Forschungsarbeiten bei Softwaregroßprojekten zeigen auf, welche zusätzlichen Herausforderungen autonome Teams bewältigen müssen. Agile Frameworks im Umfeld von Softwaregroßprojekten wie SAFe, Scrum-at-Scale, DAD, the Spotify Model oder LeSS, wurden hinsichtlich Herausforderungen und Erfolgsfaktoren von Edison et al. genauer untersucht, um einen systematischen Vergleich zu ermöglichen. Für diese fünf etablierten Frameworks konnten insgesamt 31 Herausforderungen und 27 Erfolgsfaktoren identifiziert werden [14]. Diese skalierten agilen Frameworks sind jedoch nicht die einzigen Methoden, die in Softwaregroßprojekten Anwendung finden. Einige Softwaregroßprojekte orientieren sich an hybriden Modellen. Diese kombinieren traditionelle Softwareentwicklungsprozesse mit agilen Ansätzen. Zu diesen gehört zum Beispiel das Wasserfallmodell mit Scrum, auch „Scrumfall“ genannt [1].

Da diese Modelle nicht ohne projektspezifische Anpassungen eingesetzt werden können, ist es wichtig, konkrete Umsetzungskontexte zu analysieren. Ein gutes Beispiel für eine kontextspezifische Anpassung liefert die Arbeit von Kalenda et al., die Literatur zu Praktiken, Herausforderungen und Erfolgsfaktoren in LeSS und SAFe analysierten und als Grundlage für eine aktionsorientierte Untersuchung (Engl. „action research“) bei der Firma Kentico heranzogen [49].

Neben der Frage der kontextbezogenen Anwendung rücken in der aktuellen Forschung auch strukturelle Herausforderungen wie die Koordination autonomer Teams in den Mittelpunkt. Ein zentrales Thema aktueller Forschung ist die erschwerte Koordination und Kommunikation zwischen autonomen Teams in großen Softwareprojekten. So zeigen Berntzen et al. und Bjarnason et al., dass Inter-Team-Koordination eine der größten Herausforderungen darstellt. Berntzen et al. entwickelten hierzu das sogenannte TOPS-Framework, das 27 Koordinationsmechanismen in drei Kategorien – Meetings, Roles, Tools & Artefacts – gliedert. Jeder Mechanismus wird durch eine primäre Charakteristik (Technical, Organizational, Physical oder Social) beschrieben [6] [7].

Bjarnason et al. wiederum strukturieren die Einflussfaktoren auf Inter-Team-Kommunikation in vier Gruppen: Awareness of others, Interaction between other teams, Attitude to others und Team characteristics [8].

Neben Koordination und Kommunikation beschäftigen sich weitere aktuelle Forschungsarbeiten mit den Themen Leadership, Organizational Context, Team Design und Team Processes [2] [4] [50].

Neben den Herausforderungen der Koordination und Kommunikation autonomer Teams in Softwaregroßprojekten, rücken auch Fragen zur konkreten Arbeitsweise autonomer Teams zunehmend in den Fokus: Warum und nach welchen Kriterien verteilen Teammitglieder Aufgaben auf sich selbst – und inwieweit decken sich diese Entscheidungen mit den Interessen des Managements [51]?

Best Practices auf Team-Ebene wurden bislang nur selten gezielt im Kontext von Softwaregroßprojekten untersucht. Zwar liefern große Frameworks wie SAFe oder LeSS Anhaltspunkte für die Projektplanung, doch sind diese oft sehr abstrakt und müssen an konkrete Gegebenheiten angepasst werden.

Ziel dieser Arbeit ist es daher, anhand von drei Fallstudien zu analysieren, ob sich praktikable Best Practices für autonome Teams in Softwaregroßprojekten identifizieren lassen.

Die von Hoda et al. entwickelten Balancing Acts mit ihren sieben thematischen Kategorien bieten dafür eine strukturierte und breit gefächerte Analysegrundlage. Sie ermöglichen es, zentrale Aspekte wie Entscheidungsfindung, Aufgabenzuteilung oder Wissensverteilung in autonomen Teams gezielt zu untersuchen. Durch die klare Kategorisierung lassen sich gefundene Praktiken leichter vergleichen und hinsichtlich ihrer Skalierbarkeit einordnen. Die zugehörigen Praktiken sind in Tabelle 3 schlagwortartig den Kategorien zugeordnet; ihre detaillierte Beschreibung findet sich in den Arbeiten „Self-Organizing Agile Teams: A Grounded Theory“ [24] und „Balancing Acts: Walking the Agile Tightrope“ [25] sowie im Kapitel 3.2 dieser Arbeit.

3.1 Die Basis für diese Arbeit

Wie im vorherigen Kapitel angeführt, rückt das Thema „autonome Teams in Softwaregroßprojekten“ zunehmend in den Fokus wissenschaftlicher Untersuchungen. Eine systematische Analyse der täglichen Arbeitsweise auf Teamebene steht jedoch noch aus – diese Lücke adressiert die vorliegende Arbeit anhand von drei Fallbeispielen.

Eine inhaltliche und strukturierte gute Basis für die Untersuchung der Best Practices dieser Arbeit bilden die „Self-organizing Agile Team Practices“ von Hoda et al. [25], [26]. Das Konzept der Balancing Acts bildet den theoretischen Rahmen für die vorliegende Untersuchung. Es beschreibt zentrale Spannungsfelder, die selbstständiges und selbstorganisiertes Arbeiten autonomer Teams im Alltag ermöglichen und strukturieren. In dieser Arbeit werden diese Themen im Kontext von Softwaregroßprojekten aufgegriffen und ihre Skalierbarkeit anhand empirischer Fallstudien untersucht. Die Balancing Acts umfassen dabei sieben Kategorien und 13 konkrete Praktiken, die als Grundlage für die spätere Analyse dienen.

Die Beschreibung der Balancing Acts ist wie folgt angegeben:

> *„The balancing acts include several low-level practices that enable self-organization on an everyday basis.”* ([24], S. 103)

Die Tabelle 2 zeigt die Unterteilung der drei Balancing Acts mit den zugehörigen sieben Kategorien. Zu jeder dieser Kategorien werden im nachfolgenden Kapitel 3.2 die konkreten Praktiken beschrieben.

Balancing Freedom & Responsibility	Balancing Crossfunctionality & Specialization	Balancing Continuous Learning & Iteration Pressure
• Collective decision making • Self-assignment • Self-monitoring	• Need for specialization • Encouraging cross-functionality	• Self-evaluation • Self-improvement

Tabelle 2: Die Übersicht der Balancing Acts

Als theoretische Grundlage für die Kategorie Self-assignment wird ergänzend das Paper „How Agile Teams Make Self-Assignment Work: A Grounded Theory Study“ herangezogen. Die Studie beschreibt, wie und warum sich Teammitglieder Aufgaben selbst zuweisen und welche Herausforderungen dabei entstehen [52].

Die Abbildung 6 (vgl. [24]) zeigt die Übersicht der 13 Praktiken zu den sieben Kategorien und den drei Balancing Acts. Die Praktiken dürfen nicht rein isoliert betrachtet werden, sondern haben eine gegenseitige Wechselwirkung bzw. Abhängigkeiten wie die Abbildung 7 zeigt.

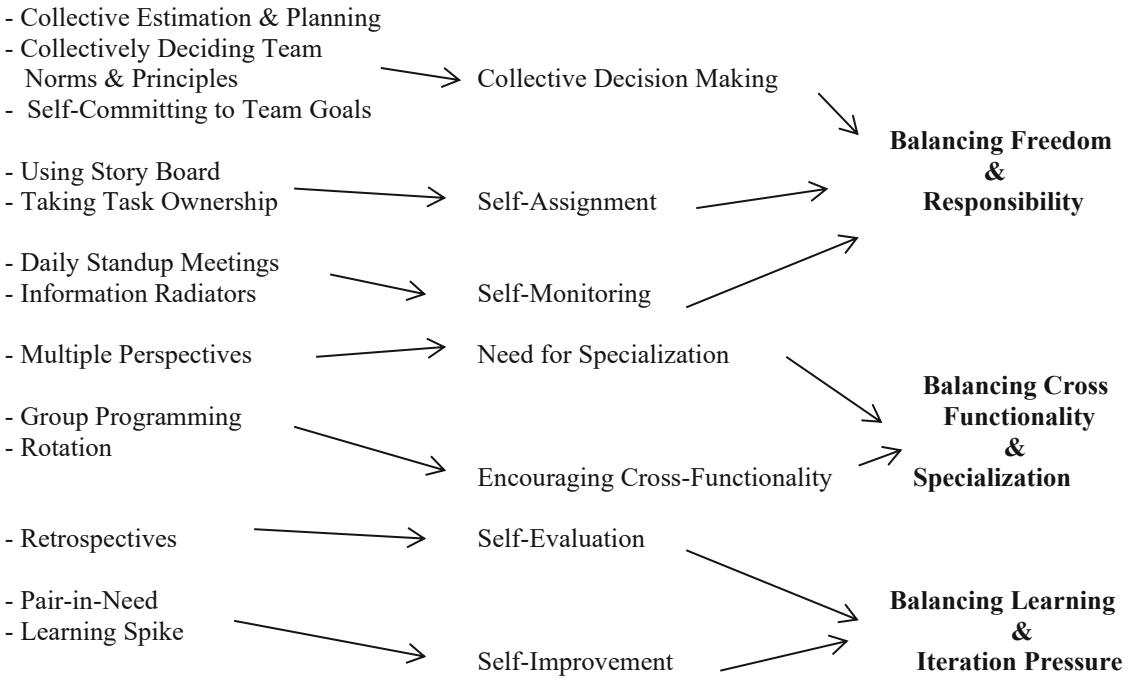


Abbildung 6: Die Balancing Acts, Kategorien und Best Practices

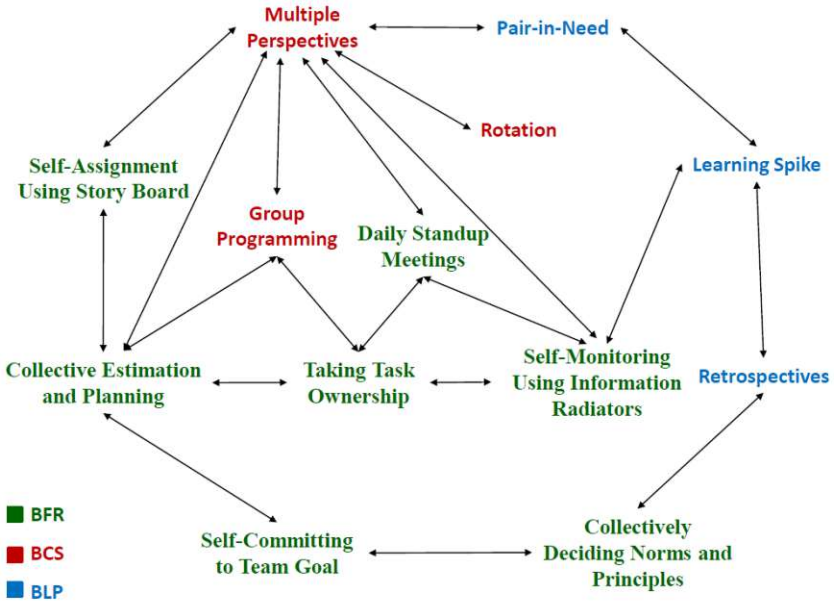


Abbildung 7: **Darstellung der Practices und deren Abhängigkeiten (Balancing Freedom & Responsibility (BFR); Balancing Cross-Functionality & Specialization (BCS); Balancing Continuous Learning & Iteration Pressure (BLP) [24]**

Eine weitere theoretische Grundlage für die Best Practices bilden die sogenannten „ABCs of Team Competencies“, ein Modell von Salas et al., das Teamkompetenzen in drei übergeordnete Bereiche gliedert:

- Einstellungen (Engl. „attitudes“)
- Verhaltensweisen (Engl. „behaviors“)
- gemeinsames Wissen bzw. mentale Modelle (Engl. „cognitions“)

Die darin enthaltenen KSAs (Engl. „knowledge, skills and abilities“) gelten als zentrale Erfolgsfaktoren für effektive Teamarbeit in komplexen Organisationsformen [53] [54].

In dieser Arbeit dienen die KSAs als konzeptionelle Grundlage, um die Ergebnisse von Hoda et al. fundierter mit konkreten Teamkompetenzen in Verbindung zu setzen. Damit leisten sie einen Beitrag zur Validierung und zum besseren Verständnis der zugrundeliegenden Anforderungen an autonome Teams.

Die Zuordnung erfolgte anhand folgender Kriterien:

- Wird im Practice ein bestimmtes Verhalten oder Prozessmuster gefordert? → Behavior
- Ist eine bestimmte Einstellung oder soziale Haltung für die Practice zentral? → Attitude
- Erfordert die Best Practice gemeinsames Wissen? → Cognition

Einige KSAs können mehreren Kategorien zugeordnet werden. Ziel ist es eine sinnvolle inhaltliche Verknüpfung abzubilden. Die KSAs dienen in dieser Arbeit als unterstützendes Analyseinstrument, um die Praktiken in ihren zugrundeliegenden Teamkompetenzen besser zu verstehen.

Die Tabelle 4 zeigt die Zuordnung dieser KSAs zu den Kategorien nach Hoda et al. [25] [26], basierend auf den Arbeiten von Salas et al. Für alle Kategorien konnte mindestens eine passende KSA identifiziert werden – mit Ausnahme von Self-Improvement, welche sich primär auf individuelles Lernen und persönliche Kompetenzentwicklung bezieht, einem Aspekt, der im ABC-Modell nicht explizit berücksichtigt wird.

Die Gegenüberstellung in der Tabelle 4 verdeutlicht, dass die von Salas et al. formulierten KSAs – publiziert 2009 und 2010 – konzeptionelle Überschneidungen mit den Balancing Acts von Hoda et al. (entstanden 2010 und 2012) aufweisen. Dies stützt die Entscheidung, das Modell von Hoda et al. als Grundlage für die Strukturierung der Interviewfragen in dieser Arbeit heranzuziehen. Dass dieses Thema nach wie vor Gegenstand aktueller Forschung ist, zeigt sich am Paper „How Agile Teams Make Self-Assignment Work: A Grounded Theory Study“ aus dem Jahr 2020.

Balancing Freedom & Responsibility	**Balancing Crossfunctionality & Specialization**	**Balancing Continuous Learning & Iteration Pressure**
Collective estimation and planning *(Collective decision making)* • Planning iterations • Commit own team goals • Estimate complexity of tasks	Multiple perspectives *(Need for specialization)* • Share and learn from each other • Team-members from different roles interact (tester, developer, …) • No strict boundaries → cross-functional • Specialized tasks before cross-functional tasks • Mature cross functional teams are highly cohesive and cooperative • Unavailability or unforeseen loss can be bypassed by other members	Self-evaluation through retrospectives *(Self-evaluation)* • Continuous learning • Respond to dynamic requirements • Learning over an iteration • Pressure to deliver every iteration (but little pressure is necessary to motivate for delivering goals)
Collectively deciding team norms and principles *(Collective decision making)*	Group Programming *(Encouraging Cross-functionality)*	Learning spike *(Self-Improvement)* • Exclusive time set aside for learning

• Collective decision making for estimation, planning, deciding team norms, principles, self-committing team goals (senior management must provide an environment)	• Direct communication (saves time) • Better understanding • Good for newcomers (support) • Expertise in other areas • Flexibility to work in different areas	• e.g. lagging behind in an area, update knowledge • Not all members must join the learning spike
Self-committing to team goals *(Collective decision making)* • Freedom to set own team goals • Ensure to achieve this goal	Rotation (Encouraging Cross-functionality) • Knowledge sharing • Keep work environment interesting because of challenging new areas	Pair-in-need *(Self-Improvement)* • Solving complex tasks in Pair-Programming
Self-assignment using storyboards *(self-assignment)* • Self-assignment of tasks (e.g. over storyboard) • Individual decisions • Transparency		
Taking task ownership *(self-assignment)* • Initiales of names or avatars as identifier • Responsibility for the task • Business priority over own technically skills or easy implementation • Avoid conflicts during self-assignment • Inexperienced teams become guidance from coaches/seniors		
Self-monitoring through daily standups and information radiators *(self-monitoring)* • Monitor progress through iteration to ensure goal achievement • … responsibility for that is shared among all members of the team • Daily standup meetings for status • Charts like Burn down charts help to visualize the progress • Peer pressure (not by manager) • Information radiators (artifacts that radiate project information)		

Tabelle 3: Praktiken der Selbstorganisation in Teams - The Balancing Acts

	KSA	ABC	Kategorie(n) nach Hoda et al.	Begründung
1	Team/Collective Orientation	Attitude	Self-assignment, Collective decision making	Fördert gemeinsame Zielausrichtung und Verantwortungsübernahme bei der Aufgabenwahl und Entscheidungsfindung
2	Team/Collective Efficacy	Attitude	Self-monitoring, Self-evaluation	Stärkt das Vertrauen in die eigene Leistungsfähigkeit des Teams bei Reflexion und Anpassung
3	Psychological Safety	Attitude	Collective decision making, Self-monitoring	Ermöglicht offenes Feedback und Fehlerkultur, fördert transparente Entscheidungen
4	Team Learning Orientation	Attitude	Self-evaluation, Self-improvement	Unterstützt kontinuierliches Lernen und Weiterentwicklung im Team
5	Team Cohesion	Attitude	Collective decision making	Stärkt den sozialen Zusammenhalt bei gemeinsamen Entscheidungen
6	Mutual Trust	Attitude	Self-assignment, Self-monitoring	Vertrauen ist essenziell für selbstständige Aufgabenübernahme und gegenseitige Kontrolle
7	Team Empowerment	Attitude	Self-assignment, Self-monitoring	Teams übernehmen Verantwortung für eigene Prozesse und Aufgaben
8	Team Reward Attitude	Attitude	Self-evaluation	Wertschätzung gemeinsamer Leistungen fördert Selbstreflexion
9	Team Goal Commitment / Conscientiousness	Attitude	Collective decision making, Self-monitoring	Hohe Zielbindung unterstützt gemeinsame Entscheidungen und Fortschrittskontrolle
10	Mutual Performance Monitoring	Behavior	Self-monitoring	Ermöglicht kontinuierliche Überwachung der Teamleistung
11	Adaptability	Behavior	Self-monitoring, Self-assignment	Flexibilität ist zentral für selbstbestimmte Aufgabenwahl und Prozessanpassung
12	Backup/ Supportive Behavior	Behavior	Self-monitoring	Rollenübergreifende Unterstützung ist Teil kollektiver Selbstkontrolle
13	Implicit Coordination Strategies	Behavior	Self-monitoring	Nicht explizite Koordination ergänzt formelle Selbststeuerung
14	Shared/ Distributed Leadership	Behavior	Self-monitoring, Collective decision making	Kollektive Führung stützt Entscheidungsfindung und Selbstorganisation
15	Mission Analysis	Behavior	Self-evaluation	Analyse der Aufgabenbasis bildet Grundlage für Reflexion und Bewertung
16	Problem Detection	Behavior	Self-monitoring	Frühe Problemidentifikation fördert effektive Selbstbeobachtung
17	Conflict Resolution/Management	Behavior	Collective decision making	Entscheidungen erfordern konstruktiven Umgang mit Meinungsverschiedenheiten
18	Motivation of Others	Behavior	Self-assignment, Self-monitoring	Gegenseitige Motivation unterstützt Selbstverantwortung und Ausdauer
19	Intrateam Feedback	Behavior	Self-monitoring, Self-evaluation	Feedback ist Schlüssel zur kontinuierlichen Verbesserung
20	Task-related Assertiveness	Behavior	Collective decision making	Klarheit in der Kommunikation fördert effiziente Entscheidungsprozesse
21	Planning	Behavior	Self-evaluation	Reflexion erfordert strukturierte Planung von Verbesserungen

22	Coordination	Behavior	Self-monitoring, Encouraging cross-functionality	Koordination ist Grundlage für reibungslosen Ablauf im interdisziplinären Team
23	Team Leadership	Behavior	Self-monitoring	Führungskompetenz ist zentral für Selbststeuerung
24	Problem Solving	Behavior	Self-monitoring, Self-evaluation	Lösungsorientierung ist Basis für Prozessanpassung und Bewertung
25	Closed-loop Communication / Information Exchange	Behavior	Self-monitoring	Rückkopplungsschleifen sichern Informationsqualität im Team
26	Cue-Strategy Associations	Cognition	Self-monitoring	Strategiewissen unterstützt situationsbezogene Entscheidungen
27	Accurate Problem Models	Cognition	Self-evaluation	Ein realistisches Problemverständnis erleichtert Bewertung und Ableitung von Maßnahmen
28	Accurate and Shared Mental Models	Cognition	Collective decision making, Encouraging cross-functionality	Geteilte mentale Modelle fördern gemeinsame Entscheidungen und rollenübergreifendes Arbeiten
29	Understanding of Team Mission, Objectives, Norms, Resources	Cognition	Self-evaluation	Verständnis der Rahmenbedingungen ist Grundlage für Bewertung
30	Understanding Multiteam Systems (MTS) Couplings	Cognition	Self-monitoring	Verständnis übergreifender Abhängigkeiten unterstützt übergreifende Selbststeuerung

Tabelle 4: Zuordnung KSAs zu den Kategorien von Hoda et al.

3.2 Kategorien und Best Practices von Hoda et al.

Dieses Kapitel führt in die sieben Kategorien und zugehörigen 13 Praktiken von Hoda et al. ein. Jede Kategorie wird mit einer kurzen Beschreibung und den zugehörigen Praktiken erläutert (vgl. [24]).

3.2.1 Collective Decision-Making

Diese Kategorie legt den Fokus auf kollaborative Entscheidungsfindung, bei der das gesamte Team in Planungs- und Entscheidungsprozesse eingebunden wird. Sie beinhaltet folgende Praktiken: *Collective Estimation & Planning, Collectively Deciding Team Norms & Principles*, und *Self-Committing to Team Goals*.

3.2.1.1 Collective Estimation & Planning

Teams planen Iterationen gemeinsam und schätzen die Komplexität von Aufgaben, um ein gemeinsames Verständnis und die Transparenz zu fördern. In der Forschung von Hoda et al. wird Planning Poker als Methode beschrieben, bei der Teammitglieder für jede Aufgabe eine Karte mit einer geschätzten Komplexität auswählen. Unterschiede werden diskutiert, bis ein Konsens erreicht wird.

3.2.1.2 Collectively Deciding Team Norms & Principles

Teams diskutieren und einigen sich auf gemeinsame Normen und Prinzipien für die Zusammenarbeit, einschließlich Arbeitszeiten und Qualitätsrichtlinien. Das Senior Management unterstützt diesen Prozess, indem es den Teams Entscheidungsfreiräume einräumt.

3.2.1.3 Self-Committing to Team Goals

Diese Praktik beschreibt, wie sich ein selbstorganisiertes Team gemeinsam auf ein selbst festgelegtes Ziel für eine Iteration verpflichtet. Im Rahmen der Planung analysiert das Team die Anforderungen, schätzt den Aufwand und entscheidet eigenständig, was im kommenden Zyklus realistisch erreicht werden kann. Das Ziel wird also nicht von außen vorgegeben, sondern entsteht durch gemeinsame Abstimmung innerhalb des Teams. Wichtig ist dabei, dass sich alle Teammitglieder aktiv hinter das vereinbarte Ziel stellen und Verantwortung für dessen Umsetzung übernehmen. Die Praktik betont damit sowohl die Freiheit des Teams, den Umfang selbst zu bestimmen, als auch die gemeinsame Verantwortung, das Ziel zu erreichen.

3.2.2 Self-Assignment

Diese Kategorie betont die Autonomie der Teammitglieder bei der Auswahl und Übernahme von Aufgaben. Sie umfasst die Praktiken: *Using Story Boards* und *Taking Task Ownership*. Selbstzuweisung wird als zentrales Element selbstorganisierender Teams verstanden, da sie sowohl Verantwortungsübernahme als auch Eigenmotivation fördert. Durch die Möglichkeit, Aufgaben entsprechend der eigenen Fähigkeiten und Interessen auszuwählen, können Teammitglieder ihre Stärken gezielter einbringen, was zu höherer Produktivität und Arbeitszufriedenheit führt.

3.2.2.1 Using Story Boards

Storyboards visualisieren Aufgaben mit den Spalten "Not Assigned," "Check-Out," und "Done". Avatare oder Initialen markieren Verantwortlichkeiten und Teammitglieder weisen sich Aufgaben selbst zu. Die visuelle Transparenz der Storyboards unterstützt nicht nur die Selbstzuweisung, sondern auch die gegenseitige Abstimmung im Team. Wie Hoda et al. beschreiben, ermöglichen diese Boards eine kollaborative Planung, fördern Diskussionen über Prioritäten und helfen, den Überblick über die individuelle wie kollektive Arbeitsbelastung zu behalten.

3.2.2.2 Taking Task Ownership

Mitglieder übernehmen Verantwortung für Aufgaben, indem sie diese eigenständig auswählen und abschließen. Unterstützung durch Coaches hilft insbesondere unerfahrenen Teams, passende Aufgaben auszuwählen. Im Sinne von Ownership führen Teammitglieder ihre Aufgaben aus und tragen Verantwortung für deren Ergebnis sowie den Projekterfolg. Aufgaben werden nicht getrennt betrachtet, sondern als Beitrag zum gemeinsamen Ziel verstanden. Treten Hindernisse auf, werden diese proaktiv adressiert oder passende Unterstützung herangezogen.

3.2.3 Self-Monitoring

Diese Kategorie zielt darauf ab, Fortschrittsüberwachung und Transparenz sicherzustellen. Sie beinhaltet die Praktiken*: Daily Standup Meetings* und *Information Radiators*. Es wird betont, dass selbstorganisierende Teams ihren Fortschritt kontinuierlich selbst reflektieren und anpassen müssen, um ihre Ziele zu erreichen. Der Austausch im Team ersetzt hierbei klassische Kontrolle von außen und stärkt zugleich die gegenseitige Verantwortung.

3.2.3.1 Daily Standup Meetings

Standups werden für Statusupdates eingesetzt. Jedes Teammitglied gibt ein kurzes Update darüber, was es am Vortag erreicht hat, was es heute plant und welche Hindernisse eventuell bestehen. Burndown-Charts, die verbleibende Arbeit visualisieren, helfen dem autonomen Team bei der Fortschrittsmessung.

3.2.3.2 Information Radiators

Artefakte wie Storyboards zeigen den Fortschritt klar und sichtbar für das gesamte Team. Typische Spalten sind "Not Assigned," "Check-Out," und "Done". Solche visuell zugänglichen Darstellungen fungieren als permanente Informationsquelle für alle Beteiligten. Sie unterstützen nicht nur die Selbstüberwachung, sondern auch die teamweite Abstimmung und fördern ein gemeinsames Bewusstsein für Prioritäten und verbleibende Aufgaben.

3.2.4 Need for Specialization

Diese Kategorie widmet sich den Rollen und Perspektiven. Sie beinhaltet die Best Practice: *Multiple Perspectives*. Die Spezialisierung ermöglicht es Teammitgliedern ihr Fachwissen gezielt einzubringen und komplexe Anforderungen zu bewältigen. Gleichzeitig bleibt die übergreifende Zusammenarbeit entscheidend, um ein umfassendes Verständnis im Team zu sichern.

3.2.4.1 Multiple Perspectives

Die Teams nutzen die Vielfalt von Rollen wie Tester, Entwickler und Analysten, um gemeinsam bessere Lösungen zu finden. Die Zusammenarbeit zwischen verschiedenen Rollen fördert den Wissensaustausch und hilft andere Ansätze und Ideen in Lösungen miteinfließen zu lassen. Dabei werden traditionelle Rollengrenzen bewusst aufgelöst, um einen echten Perspektivenaustausch zu ermöglichen. Der bewusste Einbezug unterschiedlicher Sichtweisen trägt zur besseren Entscheidungsfindung bei und erhöht die Qualität.

3.2.5 Encouraging Cross-Functionality

Diese Kategorie fördert die Zusammenarbeit und Flexibilität im Team. Sie umfasst die Praktiken: *Group Programming* und *Rotation*. Ziel ist es ein gemeinsames Verantwortungsbewusstsein für das Produkt zu schaffen. Cross-Funktionalität unterstützt eine Teamstruktur, in der Mitglieder nicht nur ihre Kernrolle ausüben, sondern flexibel auf neue Anforderungen reagieren können.

3.2.5.1 Group Programming

Offene Arbeitsumgebungen fördern den Wissensaustausch und erleichtern die (direkte) Kommunikation. Neue Mitglieder profitieren von der Unterstützung durch erfahrenere Kollegen. Das gemeinsame Arbeiten an Aufgaben – ob in Form von Pair-Programming oder informellen Gruppen

– ermöglicht nicht nur schnellere Einarbeitung, sondern auch kontinuierliche Qualitätssicherung. Fachwissen wird dabei nicht zentralisiert, sondern bewusst verteilt und gemeinsam weiterentwickelt.

3.2.5.2 Rotation

Regelmäßige Rotationen von Verantwortlichkeiten und auch von Teammitgliedern zu anderen Teams und Themen erweitern die Fähigkeiten der Mitglieder und fördern die Zusammenarbeit. Wechselnde Aufgaben und Perspektiven helfen Abhängigkeiten zu reduzieren und das Verständnis zur verbessern. Rotationen stärken die Anpassungsfähigkeit des Teams und beugen Wissensinseln aktiv vor.

3.2.6 Self-Evaluation

Diese Kategorie legt Wert auf Reflexion und kontinuierliche Verbesserung. Sie beinhaltet die Best Practice: *Retrospectives*. Selbstorganisierende Teams benötigen regelmäßige Gelegenheiten zur Selbstreflexion, um ihre Arbeitsprozesse kritisch zu hinterfragen.

3.2.6.1 Retrospectives

Teams reflektieren regelmäßig über ihre Arbeitsweise, um Verbesserungsmöglichkeiten zu identifizieren. Entscheidungen aus Retrospektiven werden genutzt, um Prozesse effizienter zu gestalten. Retrospektiven fördern eine Kultur des Lernens, in der Fehler als Entwicklungschancen betrachtet werden. Sie helfen Teams, systematisch aus Erfahrung zu lernen, Verantwortlichkeiten zu klären und das Miteinander nachhaltig zu stärken.

3.2.7 Self-Improvement

Diese Kategorie fördert die individuelle Weiterentwicklung und das Lernen im Team. Sie umfasst die Praktiken: *Pair-in-Need* und *Learning Spike*. Selbstorganisierende Teams entwickeln sich nicht nur durch Prozesse weiter, sondern auch durch gezieltes individuelles und kollektives Lernen. Gerade in dynamischen Projektkontexten ist die Fähigkeit zur schnellen Anpassung an neue Technologien und Methoden ein zentraler Erfolgsfaktor.

3.2.7.1 Pair-in-Need

Zwei Teammitglieder arbeiten zusammen, um spezifische Probleme zu lösen. Diese Praxis wird bei komplexen und designintensiven Aufgaben angewandt und fördert den Wissensaustausch. Pair-in-Need geht über klassisches Pair-Programming hinaus: Es wird situationsbezogen eingesetzt, wenn hoher Abstimmungsbedarf oder kritische Designentscheidungen bestehen. Die Methode verbessert nicht nur die Lösungsqualität, sondern ermöglicht eine direkte Weitergabe von implizitem Wissen im konkreten Kontext.

3.2.7.2 Learning Spike

Teams nehmen sich gezielt Zeit, um neue Technologien oder Konzepte zu lernen. Ein Beispiel beschreibt ein Team, das durch den unerwarteten Weggang seines einzigen Testers unter starkem Iterationsdruck stand. Mithilfe eines Learning Spikes wurde entschieden, die Testprozesse zu automatisieren. Der agile Coach unterstützte das Team dabei, den Druck zu bewältigen und sich auf neue Tools und Techniken zu konzentrieren. Mit der Unterstützung eines neuen Testers mit erweiterten Programmierkenntnissen konnte die Automatisierung deutlich vorangetrieben werden.

4 Single Case Studies

In diesem Kapitel werden die Single Case Studies beschrieben. Im Punkt 4.1 wird auf die Erstellung des Fragebogens eingegangen. Im Anschluss folgen Informationen zum Aufbau des Interviews, Details zu den Fallbeispielen sowie der Datenanalyse und Auswertung (vgl. Punkt 1.4). In der Single Case Study wird die Auswertung für jedes Fallbeispiel unabhängig der anderen Fallbeispiele vorgenommen. Die Auswertung über alle drei Fallbeispiele gemeinsam befindet sich im Kapitel 5.

4.1 Erstellung des Fragebogens

In diesem Kapitel wird die Strukturierung der Fragen für das semi-strukturierte Interview beschrieben (siehe Phase III in Kapitel 1.4.2). Für die Vorstrukturierung der Interviews werden die sieben Kategorien der Balancing Acts (siehe Punkt 3.1 und 3.2) verwendet:

1. Collective decision making
2. Self-assignment
3. Self-monitoring
4. Need for specialization
5. Encouraging cross-functionality
6. Self-evaluation
7. Self-improvement

Diese Strukturierung ermöglicht eine gezielte Analyse und den Vergleich der Interviews mit den Best Practices von Hoda et al. Ergänzend dazu erlaubt die thematische Analyse nach Braun & Clarke [23], durch offene Fragen weitere relevante Kategorien zu identifizieren.

In den Unterkapiteln 4.1.1 bis 4.1.7 werden die sieben Hauptkategorien angeführt. Der Fokus liegt auf inhaltlichen Beschreibungen, die zentrale Aspekte und typische Themenbereiche innerhalb jeder Kategorie zusammenfassen.

4.1.1 Collective decision making

Diese Kategorie widmet sich den Praktiken: *Collective estimation and planning, Collectively deciding team norms and principles* und *Self-committing to team goals* (vgl. Punkt 3.2.1). Fragen in dieser Kategorie schließen folgende Punkte mit ein:

- Die Planung von Iterationen
- Die Abschätzung der Komplexität von Anforderungen/Aufgaben (z. B. durch Punktevergabe)
- Gemeinsame Entscheidungen über Normen, Prinzipien, Regeln und Zeitmanagement
- Vorgehensweisen zur Zielerreichung
- Die Selbstverantwortung der Teammitglieder hinsichtlich ihrer Aufgaben und Ziele

4.1.2 Self-assignment

Diese Kategorie beinhaltet die Best Practices: *Self-assignment using storyboards* und *Taking task ownership* (vgl. Punkt 3.2.2). Fragen in dieser Kategorie schließen folgende Punkte mit ein:

- Die Entscheidungsfindung bei der Auswahl von Aufgaben (z. B. anhand von Prioritäten oder Spezialisierung)
- Die Sicherstellung von Transparenz innerhalb des Teams
- Die Selbstzuordnung von Aufgaben und Hilfsmitteln (z. B. Storyboards)
- Die Kennzeichnung der Aufgabenverantwortung
- Unterstützung bei der Auswahl und Zuteilung von Aufgaben

4.1.3 Self-monitoring

Fragen zu *Self-monitoring through daily standups and information radiators* (vgl. Punkt 3.2.3) widmen sich:

- Fortschrittsverfolgung durch regelmäßige Meetings (z. B. Daily Standups)
- Nutzung von Artefakten und Tools (z. B. „Burn-down Charts“, Dashboards) zur Visualisierung und Analyse des Fortschritts
- Verantwortung für die Fortschrittsbewertung innerhalb des Teams
- Soziale Kontrolle durch Teammitglieder zur Sicherstellung termingerechter Fertigstellung

4.1.4 Need for specialization

Fragen zu der Practice *Multiple Perspectives* (vgl. 3.2.4) behandeln die Punkte:

- Erfahrungsaustausch und Lernen innerhalb autonomer Teams
- Blick über die eigenen Rollen- und Projektgrenzen hinaus
- Spezialisierung der Teammitglieder und ihre Auswirkungen auf die Aufgabenverteilung
- Strategien zur Sicherstellung von Wissenstransfer und zur Vertretung bei unerwarteten Ausfällen

4.1.5 Encouraging cross-functionality

In diese Kategorie fallen *Group programming* und *Rotation* (vgl. Punkt 3.2.5). Hierbei werden Fragen zu folgenden Themen gestellt:

- Wissensverteilung und Integration neuer Teammitglieder
- Förderung von direkter Kommunikation zur schnelleren Abstimmung und Problembehebung
- Förderung von Motivation und Interesse durch abwechslungsreiche Aufgaben

4.1.6 Self-evaluation

In dieser Kategorie befindet sich die Praktik *Self-evaluation through retrospectives* (vgl. Punkt 3.2.6):

- Lernen aus vergangenen Iterationen durch Retrospektiven
- Anpassung an dynamische Anforderungen und Veränderungen
- Selbstreflexion und Weiterbildungsmöglichkeiten innerhalb des Teams

4.1.7 Self-improvement

Die letzte Kategorie behandelt zwei spezielle Best Practices *Learning spike* und *Pair-in-need* (vgl. Punkt 3.2.7):

- Zeit für gezielte Weiterbildung und Einarbeitung in komplexe Themen innerhalb von Iterationen
- Unterstützung bei komplexen Aufgaben, die nicht allein bewältigt werden können (z. B. durch Pair-Programming oder Expertenhilfe)

4.1.8 Erstellung und Aufbau des Fragebogens

Die Beobachtungen der Participant Observation aus Fallbeispiel 1 (siehe Punkt 4.4.4) bestätigen, dass die zuvor definierten Kategorien die Arbeitsweise autonomer Teams umfassend abbilden. Der detaillierte Fragebogen ist im Anhang dieser Arbeit ersichtlich.

Der Fragebogen beginnt mit einer Erfassung der Meta-Informationen. In den Regeln wird festgehalten, dass das Interview unter Einwilligung des Teilnehmers digital aufgezeichnet wird, um eine anonymisierte Transkription zu ermöglichen. Das Interview hat einen semi-strukturierten Aufbau.

Der Aufbau des Interviews gliedert sich in drei Hauptbereiche:

1. Allgemeiner Teil – Fünf Fragen zur Rolle des Teilnehmers, zur Projektart und zum -ziel, zur Team- und Projektgröße, zur hierarchischen Struktur sowie zum eingesetzten Vorgehensmodell. Diese Informationen fließen in die Fallbeispielbeschreibungen (siehe Kapitel 4.2) ein und ermöglichen eine korrekte Zuordnung.
2. Fragen zu den sieben Kategorien der Balancing Acts – Jede Kategorie enthält zwei offene Fragen, mit Ausnahme der ersten, die aus drei Fragen besteht. Die Fragen sind bewusst allgemein formuliert, um eine offene und explorative Datenerhebung zu ermöglichen und gleichzeitig potenziellen Interviewer-Bias zu reduzieren.
3. Zusatzfrage – Sie gibt den Teilnehmern die Möglichkeit, weitere relevante Informationen außerhalb der vorgegebenen Struktur zu teilen.

Der Erstentwurf des Fragebogens wurde während der Participant Observation im Fallbeispiel 1 entwickelt und mit einem Projektteilnehmer getestet. Auf Basis dieser Erfahrungen erfolgte anschließend eine Verfeinerung des Fragebogens.

4.2 Übersicht der Fallbeispiele

In diesem Kapitel wird ein Überblick über die für die Analyse herangezogenen Fallbeispiele gegeben. Die Fallbeispiele wurden so ausgewählt, dass sie hinsichtlich ihrer Größe miteinander vergleichbar sind. Detaillierte Beschreibungen finden sich in den Kapiteln 4.4 bis 4.6.

Beim Fallbeispiel 1 wurde eine Participant Observation (siehe Punkt II. in Kapitel 1.4.2) ermöglicht. Durch den Austausch mit den Mitarbeitern und dem Zugang zu deren Entwicklungswerkzeugen kann dieses Fallbeispiel detaillierter beschrieben werden als die anderen beiden Fallbeispiele. In den zwei Fallbeispielen 2 und 3 wurden leider keine Unterlagen zur Verfügung gestellt,

es konnten jedoch ausreichend beschreibende Informationen im Interview erhoben werden. Namentliche Nennungen und konkrete projektspezifische Details wurden anonymisiert.

Folgende Tabelle 5 gibt eine Übersicht über die wichtigsten Daten der drei Fallbeispiele wie Projektgröße und Teamgröße in Personen, Anzahl der Teams, Dauer, Vorgehensmodell und Bereich:

	Projekt-größe	Team-größe	Anzahl Teams	Dauer	Vorgehensmodell	Bereich
Fallbeispiel 1	~ 150	2-12	> 12	2010-laufend (2025)	Projektweit: Wasserfall Gruppenintern: Kanban	Versicherungsträger
Fallbeispiel 2	~ 170	10-15	> 11	2014 – laufend (2025)	Projektweit: Scrum Gruppenintern: Scrum & Kanban	Autohersteller
Fallbeispiel 3	~ 200	8-10	> 20	2006 - laufend (2025)	Wasserfall, Scrum & Kanban	Gesundheitsbereich

Tabelle 5: Übersicht der drei Fallbeispiele

4.3 Interviews

Die Teilnehmer der Interviews wurden vom Ansprechpartner des jeweiligen Fallbeispiels vermittelt. Das Ziel ist es, verschiedene Rollen innerhalb der autonomen Teams über alle drei Fallbeispiele zu befragen, um deren Arbeitsweisen und Sichtweisen erheben und abgleichen zu können. Zu jedem Fallbeispiel wurden fünf Teilnehmer interviewt. Einige Teilnehmer haben Doppelfunktionen innerhalb des Teams. Die Tabelle 6 gibt eine Übersicht der Rollen für die Interviews je nach Fallbeispiel. Die Teilnehmer im Fallbeispiel 1 werden mit T1 bis T5 angeführt, die Teilnehmer im Fallbeispiel 2 mit T6 bis T10 und jene im Fallbeispiel 3 mit T11 bis T15.

	Fallbeispiel 1	Fallbeispiel 2	Fallbeispiel 3
Teil-/Projektleiter			T13, T14
Requirements Engineer	T1	T9	T12
Deployment/Devops		T7	
Technischer Architekt		T6, T8	T12, T15
Test/Testmanagement	T4, T1	T10	
Teamleiter	T3, T5	T9	T11
Softwareentwickler	T1, T2, T4	T8	T15

Tabelle 6: Übersicht Teilnehmer und Rollen der Interviews

Die Teilnehmer T1 und T2 konnten im Fallbeispiel 1 direkt am Standort des Projekts befragt werden. Die anderen Teilnehmer (T3 bis T15) wurden per Videoanruf interviewt. Alle Interviews wurden mit dem Einverständnis der Teilnehmer aufgenommen, damit diese für die weitere Datenanalyse bereitstehen.

4.4 Single Case Study: Versicherungsträger (Fallbeispiel 1)

Beim ersten Fallbeispiel (F1) geht es um die Realisierung einer zukunftsorientierten, prozessoptimierten Anwendung im Versicherungsbereich, welche die Geschäftsprozesse des Kerngeschäfts dieser Institution serviceorientiert unterstützt. Mit einer geplanten Projektdauer von mehr als 10 Jahren und mit mehr als 100 internen und externen Teilnehmern sowie mehrere Institutionen, die diese Applikation nutzen, kann dieses Projekt als Softwaregroßprojekt lt. Punkt 2.2.3 bezeichnet werden. Die Arbeit in autonomen Teams erfolgt in einem komplexen Umfeld mit zahlreichen voneinander abhängigen Subsystemen, zentralisierten Tools und strikten Schnittstellenvorgaben.

4.4.1 Projektziel

Nach einer mehrmonatigen Konzeptionsphase wurde festgestellt, dass die Geschäftsprozesse der beteiligten Institutionen in acht fachliche Teilbereiche gegliedert sind – u. a. Leistungen, Schriftverkehr, Rechtsangelegenheiten und medizinische Belange. Zwar existieren bereits EDV-gestützte Anwendungen, diese sind jedoch auf einzelne Fachbereiche beschränkt. Dadurch bestehen für gleichartige Aufgaben unterschiedliche, fachbereichsspezifische Lösungen, was zu einer Vielzahl an Schnittstellen führt, um Daten synchron zu halten.

Ziel des Projekts ist die durchgängige Optimierung und Harmonisierung aller Geschäftsprozesse durch eine integrierte Softwarelösung. Diese soll strategische, wirtschaftliche, fachliche und technische Anforderungen erfüllen und eine rollenbasierte Oberfläche mit einheitlichem Look & Feel bereitstellen. Technisch werden zentrale Datenhaltung, einheitliche Schnittstellen und Strukturen sowie die Reduktion von Redundanz und die Wiederverwendbarkeit von Komponenten angestrebt.

Die Projektorganisation (vgl. Abbildung 8) besteht aus zwei Hauptstrukturen: einer fachlich-organisatorischen Leitung und einer technischen Leitung (Farbe Gelb). Die Koordination erfolgt über Assistenzfunktionen sowie fachliche und technische Koordinationsrollen (Farbe Blau).

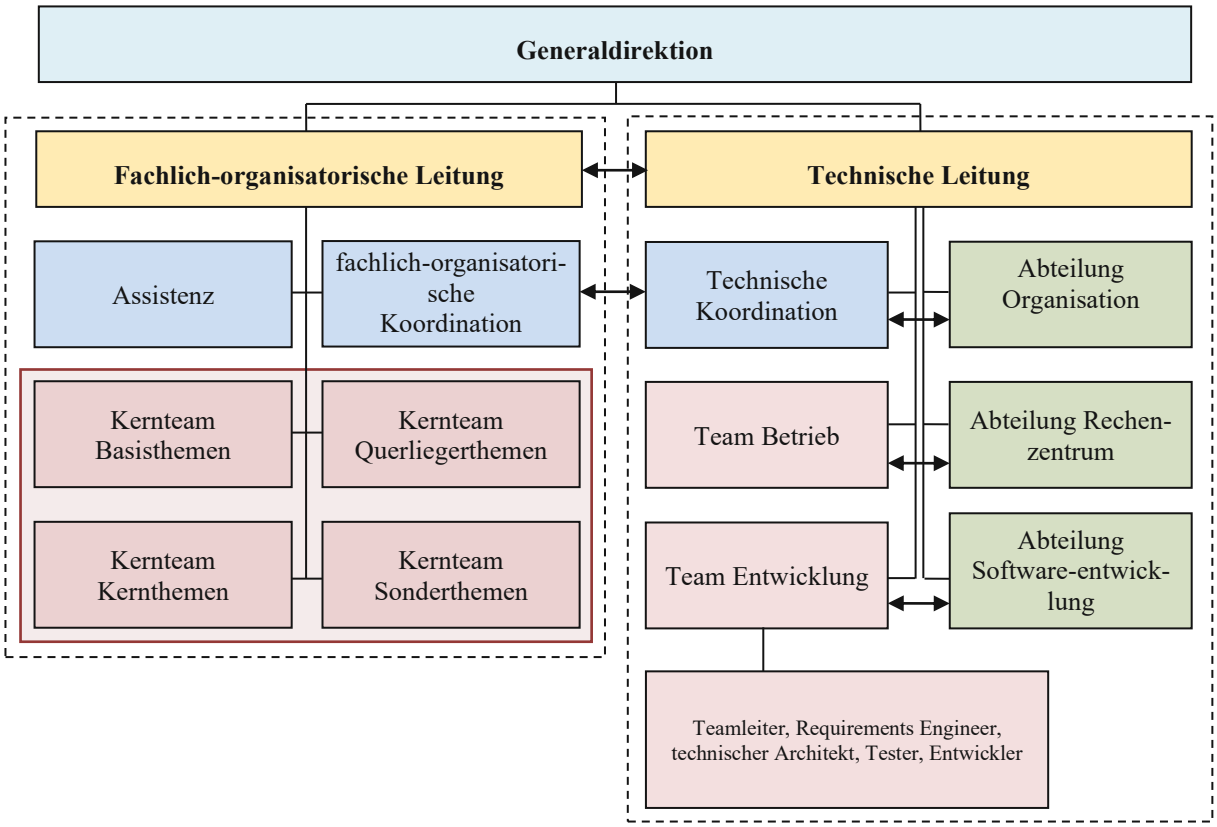


Abbildung 8: Projektorganisationsstruktur im Fallbeispiel 1

Zur fachlich-organisatorischen Koordination zählen der Testmanager, Servicemanager, Business Architekt, Projektorganisator, das Programm-Management, internes Controlling sowie Ansprechpartner für die nutzenden Institutionen. Die technische Koordination erfolgt durch den Enterprise Architekten sowie das Informationssicherheits- und Risikomanagement.

Die vier fachlichen Kernteams sind thematisch gegliedert in Basisthemen (z. B. IDM, Statistik, E-Government), Querliegerthemen (z. B. Druck, Archiv, Scan), Kernthemen (Haupt- und Subprozesse des Tagesgeschäfts) sowie Sonderthemen (z. B. Beratung, Steuern, rechtliche Belange). Auf technischer Seite verantwortet das Team Entwicklung die Umsetzung der fachlichen Anforderungen, inklusive Datenmodellierung, Service-Entwicklung, UI-Erstellung sowie Tests. Das Team Betrieb stellt die Infrastruktur bereit, überwacht Performance und behebt technische Störungen. Die Kernteams sind in zahlreiche kleinere Teilteams mit jeweils 2 bis 12 Personen untergliedert. Beispiele sind die Teams für Prozessmodellierung, Objektmodellierung, Schriftverkehr und Servicemodellierung. Auch die technischen Teams arbeiten in autonomen Strukturen und übernehmen sowohl fachlich-funktionale als auch generisch-technische Aufgaben wie Bibliotheksverwaltung, Updates, Migrationen oder Security.

Die Projektrollen sind vielfältig. Zu den zentralen Rollen zählen Auftraggeber, Auftragnehmer, Steuerungsausschuss, Projektleitung, (Kern-)Teamleitungen, Projektunterstützung, (Kern-)teams

sowie externes Projektcontrolling. Weitere Rollen ergeben sich projektspezifisch im Verlauf. Eine Übersicht bietet Tabelle 7:

Rolle	Projekt-leitung	Projekt-unterstützung	Team					
			Prozess-modellierung	Objekt-modellierung	Service-modellierung	Schriftverkehr	Entwicklung	Betrieb
Projektleitung	X							
fachliche Projektleitung	X							
Stv. der Projektleitung	X			X				
(Kern-)Teamleiter			X	X	X	X	X	X
Assistenz		X						
Projektorganisator		X						
Internes Projektcontrolling		X						
Programm-Management		X						
Business Architekt		X						
Enterprise Architekt		X						
Informationssicherheits- und Risikomanager		X						
Testmanager		X						
Partnerträgeransprechpartner		X						
Prozessmodellierer			X					
Prozessablaufsteuerer			X					
Servicemanager		X			-			
Servicemodellierer					X	X		
Objektmodellierer				X				
Fachexperte			X	X	X	X		
Fachanwender			X	X	X	X		
Anwendungsarchitekt							X	
GUI-Designer							X	
Database Modeler							X	
Business Unit- und Life Cycle-Manager							X	
Systemintegrator							X	
Service-Developer							X	
Infrastruktur Architekt								X
Infrastruktur-Manager								X
Server-Verantwortlicher								X
Netzwerk-Verantwortlicher								X
Storage-Verantwortlicher								X
Database-Verantwortlicher								X
System Integrations-Manager								X
Enterprise Servicebus-Verantwortlicher								X
End to End Monitoring-Manager								X
Operation								X
Verantwortlicher CMDB								X

Tabelle 7: Auszug der Rollen Fallbeispiel 1

An dieser Stelle sei noch erwähnt, dass es neben der Hauptinstitution noch vier weitere Einrichtungen gibt, welche die Applikation für sich nutzen wollen. Ein Steuerungsausschuss dient für die strategische und dispositive Lenkung. Die Projektmitarbeiteranzahl wird mit bis zu 150 Teilnehmern angegeben.

4.4.2 Vorgehensmodell und Projektphasen

Das Vorgehensmodell beim Fallbeispiel 1 entspricht dem Wasserfallmodell (vgl. Punkt 2.2.5.3). Die Phasen sind in Abbildung 9 dargestellt.

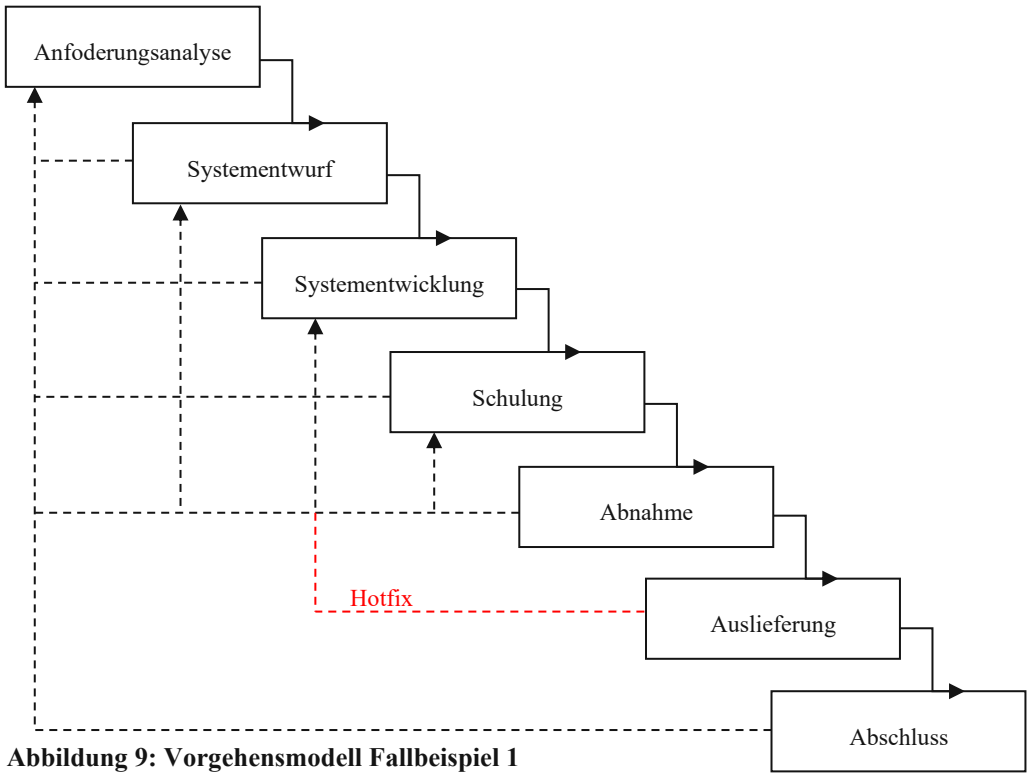


Abbildung 9: Vorgehensmodell Fallbeispiel 1

In der Phase der Anforderungsanalyse werden die Funktionalitäten sowie die notwendigen organisatorischen Voraussetzungen für den Einsatz und Betrieb definiert. Beim Systementwurf werden Entscheidungen bezüglich Systemarchitektur, Schnittstellen, Komponenten und die Vorbereitung für Tests getroffen. In der Systementwicklung findet die Umsetzung des Servicekatalogs statt. Dazu gehören alle fachlichen und technischen Services, Anbindung vorhandener Systeme, jegliche Applikationslogik, das Userinterface und die Datenablage (z.B. Datenbank). Wurden die Anforderungen umgesetzt, folgt die Phase der Schulung. Anwender und Testpersonen werden in die umgesetzten Features eingeführt. In der Abnahme erfolgen die Qualitätssicherung und das Testen. Sofern diese Phase bestanden wurde, kommt die Phase der Auslieferung. Die freigegebenen Komponenten werden in einem sogenannten „Releasepaket“ in die Produktionsumgebung

übernommen. Die letzte Phase bildet der Abschluss, in welchem ein Abschlussbericht über das (teil-)entwickelte Produkt durch die Projektleitung erfolgt.

4.4.3 Zeitplan

Der Masterplan des Fallbeispiels 1 sieht eine Projektdauer von 10 Jahren vor, beginnend Anfang 2010 bis Anfang 2020. Das gesamte Projekt ist in fünf Teilprojekte unterteilt, von denen einzelne Meilensteine für die Produktivsetzung vorgesehen sind. Es gibt somit bereits während der gesamten Laufzeit immer wieder Teilfunktionalitäten die freigeschalten werden. Teilprojekte die zusammenhängen werden, gemeinsam vorbereitet um Synergieeffekte zu erzielen, auch wenn gewisse Themen erst für spätere Teilprojekte geplant sind. Nebenbei gibt es neben der Hauptfunktionalität des Kernprojekts noch weitere interne Applikationen die parallel dazu entwickelt werden. Aufgrund weiterer, auch gesetzlich-bedingten Änderungen oder EU-Vorgaben läuft das Projekt auch aktuell (Stand 2025) noch weiter.

4.4.4 Participant Observation

Der Autor dieser Arbeit durfte über einen Zeitraum von mehreren Monaten eine Participant Observation (siehe Punkt II. in Kapitel 1.4.2) im Fallbeispiel 1 durchführen. Durch die persönliche Teilnahme konnte in diesem Zeitraum ein gutes Verständnis der Arbeitsweise und des täglichen Ablaufs der autonomen Teams in diesem Projekt erreicht werden. Neben der Beobachtung der Arbeitsweise der autonomen Teams und der Teilnahme an Meetings, bekam der Autor den Zugang zu Tools und Artefakten, wie Mockups, Anforderungsdokumente, Ticketing-System, Gitlab, SVN, Splunk (Auswertung von Logs), Objektmodelle, xWiki, etc. Im nächsten Punkt werden Beispiele mit Abbildungen der Artefakte angeführt.

4.4.5 Beispiele (Auswahl) der Artefakte und Beobachtungen

Bei Teams mit kleineren Themengebieten, deren Mitglieder oft vor Ort sind werden Whiteboards mit „Post-its“ eingesetzt. Abbildung 10 zeigt ein Beispiel für ein kleineres Themengebiet, in welchem ein Kanban-Board für „Cleanup“-Aufgaben erstellt wurde.

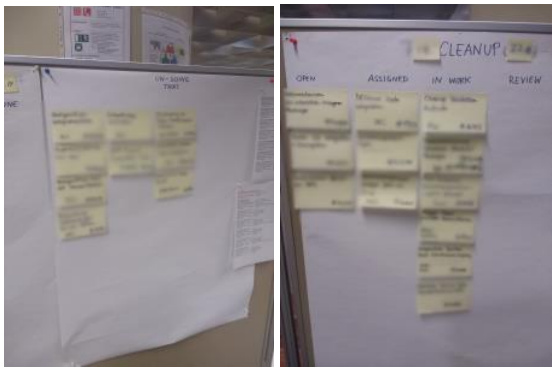


Abbildung 10: Beispiel Kanban Board (Whiteboard) Fallbeispiel 1

Die Verwendung von physischen Whiteboards ist jedoch die Ausnahme. Die meisten Themengebiete sind sehr groß und die Teammitglieder wechseln zwischen Home-Office und Büropräsenz. Da ein digitales Board leichter zu warten, übersichtlicher und transparenter ist, wird ein Kanban-Board in Gitlab eingesetzt. Dieses zeigt Abbildung 11:

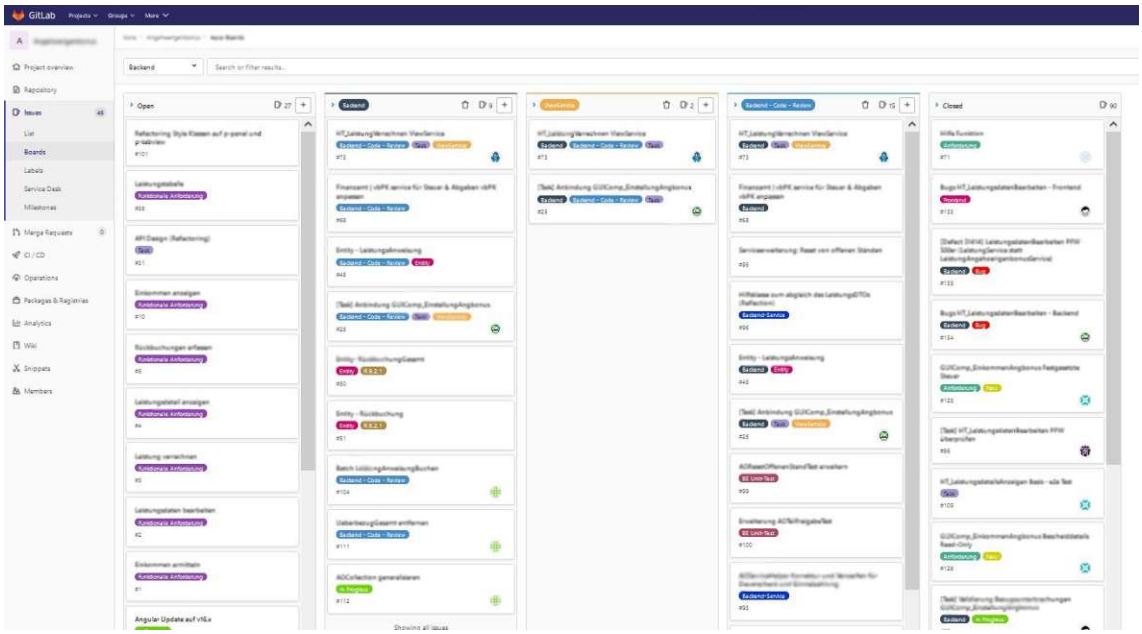


Abbildung 11: Beispiel Kanban Board Fallbeispiel 1 Gitlab

Der Vorteil der digitalen Version ist die bessere Dokumentation, die Möglichkeit Kommentare und Anhänge zu den Tasks hinzuzufügen, Filtermöglichkeiten, Vergabe von zusätzlichen Labels, eine schnellere Zuweisung usw. In den Tickets werden zum Beispiel auch die fachlichen Dokumente (Beschreibungen), GUI-Mockups, Objektmodelle (Datenbanktabellen, ER-Diagramme, etc.) oder Fehler verlinkt. Die Ablage der Mockups und Beschreibungen für Services, GUI, Batchabläufe, etc. werden in der Dokumentenverwaltungssoftware Alfresco abgelegt (siehe Abbildung 12).

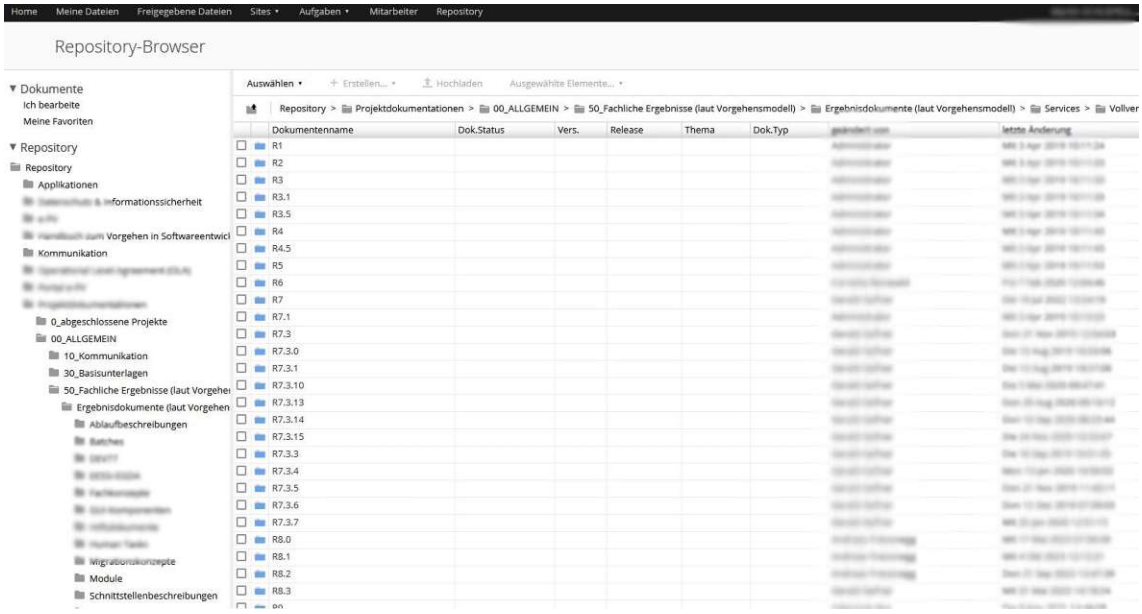


Abbildung 12: Alfresco – Beispiel Ablagestruktur nach Releases

Die Erfassung von Anforderungen und Fehlern erfolgt in der Software HPQC. Die Einteilung der Tickets erfolgt in „leicht“, „mittelt“, „schwer“ und „kritisch“. Zu jedem Ticket gibt es eine Nummer, Beschreibung, Stage, Datum, Version und Verantwortlichen. Der Status des Tickets kann „New“, „Defect Assigned“, „Rejected“, „Fixed“, „Closed”, „Reopened”, „Fachliche Analyse”, „Test”, etc. sein.

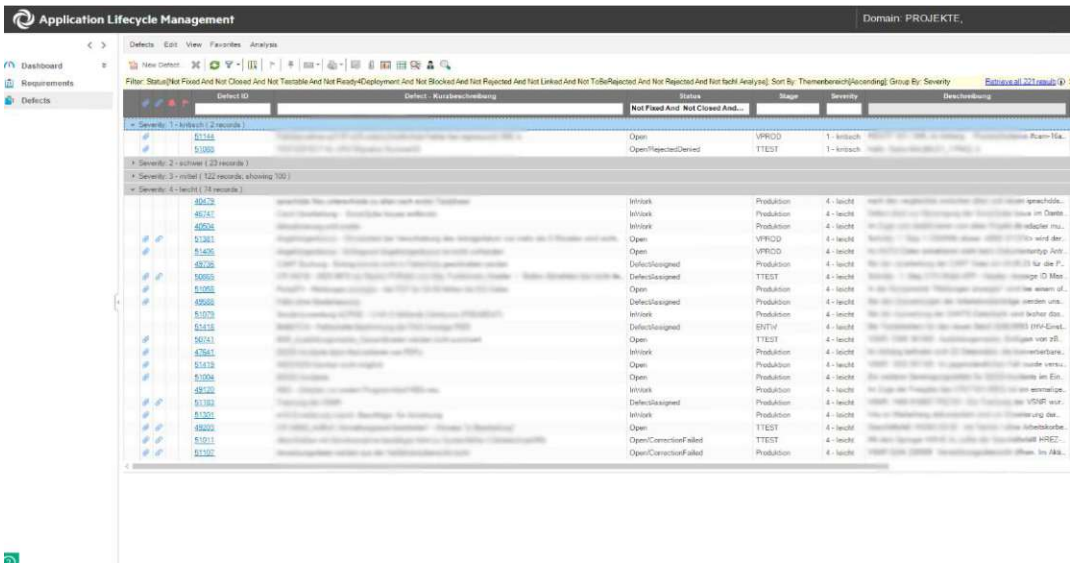


Abbildung 13: HPQC – Ticketsystem für Anforderungen und Fehler

4.4.6 Einfluss der Beobachtungen auf den Fragebogen

Da der Autor die sieben Untergruppen der Balancing Acts als Basis für den Fragebogen gewählt hat, ist es wichtig validieren zu können, ob diese Gruppen ausreichend für die Erstellung des Fragebogens und der verbundenen Datenerhebung sind und alle Teilbereiche vollständig abdecken. Durch die Beobachtung und Teilnahme in dem Projekt konnte dies bestätigt werden.

4.4.7 Thematische Analyse der Interviews

Das Vorgehen der thematischen Analyse ist für alle drei Fallbeispiele gleich und wird in diesem Kapitel anhand eines Beispiels angeführt.

Bevor die thematische Analyse nach Braun & Clarke erfolgt, werden die im Punkt 4.3 geführten aufgenommenen Interviews transkribiert. Die Erfahrungen und Artefakte der Participant Observation des Fallbeispiels 1 konnten bei den Interviews und beim Transkribieren helfen ein besseres Verständnis für Fachbegriffe und Vorgänge zu bekommen. Da diese Fachbegriffe und Vorgänge auch in den anderen beiden Fallbeispielen erwähnt werden, konnte die Participant Observation einen zusätzlichen positiven Beitrag zu dieser Phase (siehe Punkt IV. in Kapitel 1.4.2) leisten. Die Analyse gliedert sich in zwei Schritte: Im ersten Schritt, erfolgt die Codierung aller Interviews und die Bildung von Themen. Dieser Teil wird in den folgenden Punkten genauer beschrieben. Im zweiten Schritt (siehe Kapitel 5) wird mit einer Multiple Case Study über die drei Fallbeispiele nach Überschneidungen der Codes gesucht. Codes, welche sich in allen drei Fallstudien überschneiden werden als relevant eingestuft in das finale Ergebnis übernommen.

4.4.7.1 Code- und Themenbildung anhand eines Beispiels

Wie bereits in Kapitel 1.4.2 unter Punkt V angeführt bilden die transkribierten, semi-strukturierten Interviews, die Basis für die thematische Analyse.

Die thematische Analyse ist ein iterativer Prozess, bei dem Codes von Interview zu Interview verfeinert, umstrukturiert und angepasst werden. Dabei durchläuft die Analyse mehrere Schritte:

1. **Erstellung einer Dokumentengruppe** – Die transkribierten Interviews werden in MAXQDA [55] als separate Dokumente in einer neuen Gruppe gespeichert (siehe Abbildung 14).

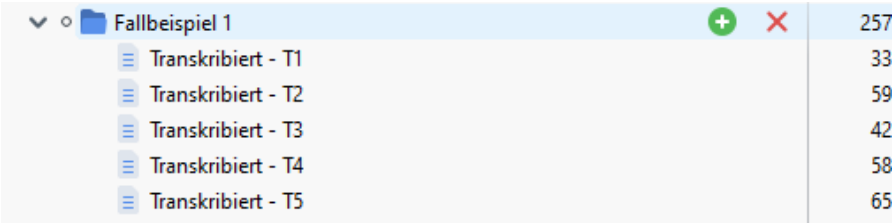


Abbildung 14: Dokumentengruppe für F1 in MAXQDA

2. **Initiale Codierung** – Wichtige Phrasen, Schlagwörter oder Absätze werden mit farblich gekennzeichneten Codes markiert (siehe Abbildung 15).

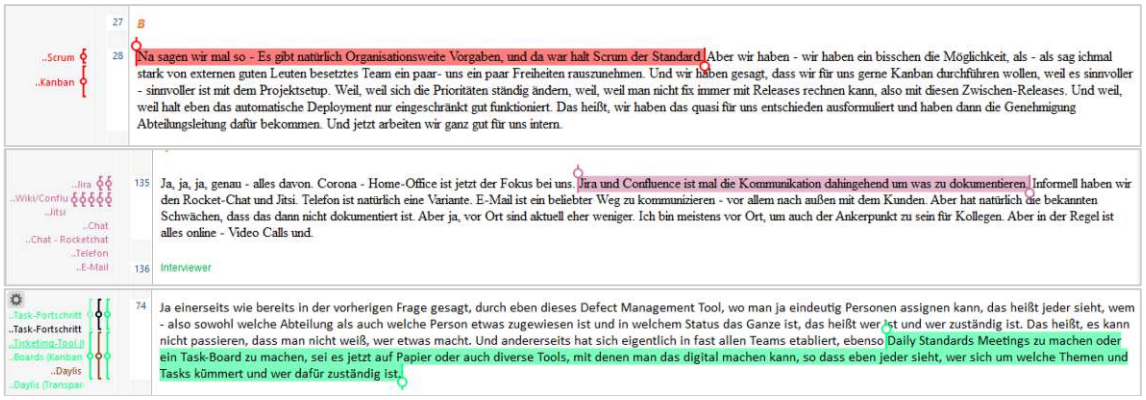


Abbildung 15: Codierung der Interviews in MAXQDA

3. **Strukturierung der Codes** – Ähnliche Codes werden zu übergeordneten Themen zusammengefasst. Dies geschieht auf Basis:
   - der Interviewstruktur (ähnliche Fragen = ähnliche Codes),
   - der Antworten mehrerer Teilnehmer (Erkennen von Redundanzen und Gemeinsamkeiten).

4. **Zusammenführung in Themen** – Codes mit inhaltlichen Überschneidungen werden zu logischen Themen gruppiert, um eine strukturierte Analyse zu ermöglichen.

Um diesen Prozess zu veranschaulichen, folgt nun ein Beispiel aus Fallbeispiel 1. Die Interviewfrage lautet:

> *„Wie erfolgt die Auswahl und Zuteilung von Anforderungen und Fehlern?“*

Ein Teamleiter (T5) antwortet im Fallbeispiel 1:

> *„Prinzipiell also jedes Team hat prinzipiell die Themen, für die es verantwortlich ist. Das heißt, jeder weiß, woran sein oder sollte wissen, woran sein Team arbeitet.*

> ***Die Zuteilung erfolgt durch die Abteilungsleitung oder durch die Abteilungs- Organisatoren**, das heißt, dort wird entschieden, welches Team ein neues Thema bekommt viel oder welches Team eben einen Defect bekommt, weil der zu einem Team dazugehört. **Im Team ist die Aufteilung im Team selbst überlassen, d. h. entweder derjenige, der sich am besten damit auskennt, im Idealfall derjenige, der es gemacht hat, oder halt derjenige, der gerade Zeit hat**, falls eben auf Urlaub ist. Wichtig ist halt, dass es so ist, dass möglichst wenig liegenbleibt."* – T5, Teamleiter

Aus dieser Antwort werden relevante Schlagwörter und Phrasen identifiziert:

1. Zuteilung durch die Abteilungsleitung
2. Zuteilung durch Organisatoren
3. Aufteilung innerhalb des Teams nach Fachwissen
4. Aufteilung nach Verfügbarkeit der Teammitglieder

Anschließend werden diese nochmals verfeinert und zusammengefasst. Die Zuteilung durch die Abteilungsleitung oder Organisatoren wird als Leitung abstrahiert.

I. Zuteilung der Leitung an das Team
II. Zuteilung/Auswahl nach Fachwissen
III. Zuteilung/Auswahl nach Ressourcen (Verfügbarkeit)

Ein anderer Teamleiter (T3) antwortet im Fallbeispiel auf dieselbe Frage:

> *„Im Prinzip haben wir intern, so teamintern **eine Übersicht über alle Themen und pro Thema haben wir eine Übersicht über alle zu erledigenden Aufgaben**. Und prinzipiell **hat jeder Zugriff drauf und sieht, wer was macht** und kann sich dann, auch wenn er in dem Thema selber drinnen ist, **selber eine Aufgabe nehmen** und hat damit auch die Einsicht."* – T3, Teamleiter

Daraus lassen sich folgende Codes ableiten:

IV. Auswahl durch Teammitglieder aus Task-Liste
V. Task-Fortschritt

Nun erfolgt die Zusammenführung in übergeordnete Themen:
Die Codes I bis IV beziehen sich auf verschiedene Formen der Aufgabenverteilung und werden dem Thema „Auswahl und Zuteilung von Tasks" zugeordnet.

Code V verweist auf Transparenz (alle sehen, wer woran arbeitet) und Fortschrittsmessung (Bearbeitung eines Tasks ist sichtbar) – er wird daher beiden Themen zugeordnet.

Nach diesem Prozess ergibt sich jene Struktur, die Tabelle 8 zeigt:

Code	Thema
I. Zuteilung der Leitung an das Team	Auswahl und Zuteilung von Tasks
II. Zuteilung/Auswahl nach Fachwissen	
III. Zuteilung/Auswahl nach Ressourcen (Verfügbarkeit)	
IV. Auswahl durch Teammitglieder aus Task-Liste	
V. Task-Fortschritt	Transparenz, Fortschrittsmessung

Tabelle 8: Beispieltabelle für die Code- und Themenbildung

Mit zunehmender Anzahl analysierter Aussagen verfeinern sich die Codes und Themen kontinuierlich. Im Laufe des Analyseprozesses kristallisieren sich konsistente thematische Gruppen heraus, die anschließend als Basis für die Kategorienbildung dienen.

4.4.7.2 Ergebnis Single Case Study Fallbeispiel 1

Die thematische Analyse für Fallbeispiel 1 führt zur Identifikation von 14 übergeordneten Themen und 66 zugehörigen Codes. Diese Ergebnisse werden aus den transkribierten Interviews abgeleitet und durch die iterative Codierung strukturiert. Die Tabelle 9 zeigt die Themen und Codes:

Anzahl der Interviews			5
Thema	**Code**	**%**	**# Dokumente**
Taskänderungen	Validierung und Einplanung nach Dringlichkeit	20%	1
	Direkt wenn nicht zu kritisch (aufgrund von Bürokratie)	80%	4
Richtlinien, Grenzen, Normen oder Prinzipien	SonarQube	40%	2
	Organisationsweite Richtlinien	80%	4
	Checkstyle	40%	2
	Zentrale Festlegung durch Spezialisten (1...n)	80%	4
	Doku (Wiki/Confluence/...)	80%	4
	Unit-Tests	20%	1
	Review nach Checkliste	40%	2
	zusätzliche Codequalitäten durch Team bzw. Teamleiter	40%	2
Meetings	Meetings bei Bedarf	40%	2
	Weekly (1-2 Wochen Zyklus)	60%	3
	Meetings klein halten	20%	1
	Dailies	80%	4

	Teamleiter-Meetings (Test, Architektur, etc.)	60%	3
Umgebung	Vorort (Kleinbüro, ...)	80%	4
Team Events & Belohnungen	Allgemeine Teamevents	80%	4
Komplexe Themen	Support bei lizensierter Software	20%	1
	4 Augen Prinzip & Reviews	100%	5
	Teamabsprache und Validierung	80%	4
	Testabdeckung um Komplexität abzudecken	20%	1
Fortschrittsmes-sung	Task-Fortschritt (Monitoring) per Board	80%	4
	Agiler Prozess	20%	1
	Leistungsfortschrittsbericht	80%	4
	wöchentliche Meetings mit Leitung	20%	1
	Ticketstatus (%, Stunden oder Status)	80%	4
Auswahl und Zu-teilung Tasks	Leitung an Team	100%	5
	Zuteilung nach Fachwissen/jenen der sich am besten aus-kennt	60%	6
	Auswahl durch Teammitglieder aus Task-Liste	100%	5
Fortbildung	Teamwechsel	40%	2
	Zertifikate & Weiterbildung (aktiv)	20%	1
	Zeit für Verbesserungen	20%	1
	Themenwechsel	60%	3
	Rollenwechsel	60%	3
	Zeit Technologiewechsel & Updates	20%	1
Transparenz	Software & Ticketverwaltung	80%	4
	Agiler Prozess	20%	1
	Task-Fortschritt für jedes Mitglied	60%	3
	Status setzen	60%	3
	Direkte Kommunikation	20%	1
	Dailies (Transparenz)	60%	3
	Boards (Kanban od. ähnliches)	100%	5
Wissensverteilung & Support	Code Review	100%	5
	JavaDoc (Code-Kommentare)	20%	1
	Dokumentation	20%	1
	Taskzuteilung nach belieben	40%	2
	Übergabemeetings	40%	2
	Direkte Nachfrage (Sms, Telefon)	20%	1
	Taskrotation	20%	1
	Zuteilung an Andere bei Leerlauf	20%	1
	Pair-Programming	40%	2
	Full-Stack Entwicklung	20%	1
	Mehrere Personen pro Thema	60%	3
Reflexion & Lern-prozess	Retrospektive & Review	20%	1
	Direkt wenn es auftritt	60%	3

Aufwandsschätzung	Schätzung über mehrere Iterationen	20%	1
	Durch Teammitglieder	100%	5
	Aufteilung in Subtasks	20%	1
	projektspezifische Einheitenschätzung	40%	2
Kommunikation	Telefon	60%	3
	direkte Kommunikation	100%	5
	Ticket-Tool	40%	2
	Chat	80%	4
	Meetings	40%	2
	E-Mail	100%	5
	Wiki/Confluence/...	20%	1

Tabelle 9: Vorkommen der 66 Codes im Fallbeispiel 1

4.5 Single Case Study: Autohaus (Fallbeispiel 2)

Beim Fallbeispiel 2 handelt es sich um einen Automobilhändler der ein neues Dealer-Management-System (DMS) implementiert. Darunter versteht man die Verwaltung von Werkstattaufträgen, Käufe von Neu- und Gebrauchtwägen inklusive Datawarehouse mit Aufbereitung von Kennzahlen und Statistiken. Die Software hat nicht nur viele interne Schnittstellen, sondern muss auch mit externen Systemen wie zum Beispiel Versicherungen und Banken kommunizieren.

4.5.1 Projektziel

Das Projektziel ist die Ablöse und Neuimplementierung eines bestehenden Dealer-Management-Systems. Aktuelle Vertragspartner in den Werkstätten und Autohäusern benutzen ein Fat-Client System, welches mit C++ implementiert ist. Dieses soll mit einer modernen Neuentwicklung mit Web-User Interfaces in Angular und einem verteilten System mit mehreren Applikationen, die miteinander kommunizieren abgelöst werden. Das Ziel ist es, dass diese Applikation nicht nur regional, sondern weltweit ausgerollt werden kann.

4.5.2 Projektorganisationsstruktur, Rollen und Teams

Die Abbildung 16 zeigt die Projektorganisationsstruktur des Fallbeispiels 2. Innerhalb des Teams gibt es mehrere Tester und Entwickler, sowie einen Teamleiter, technischen Architekten und einen Requirements Engineer.

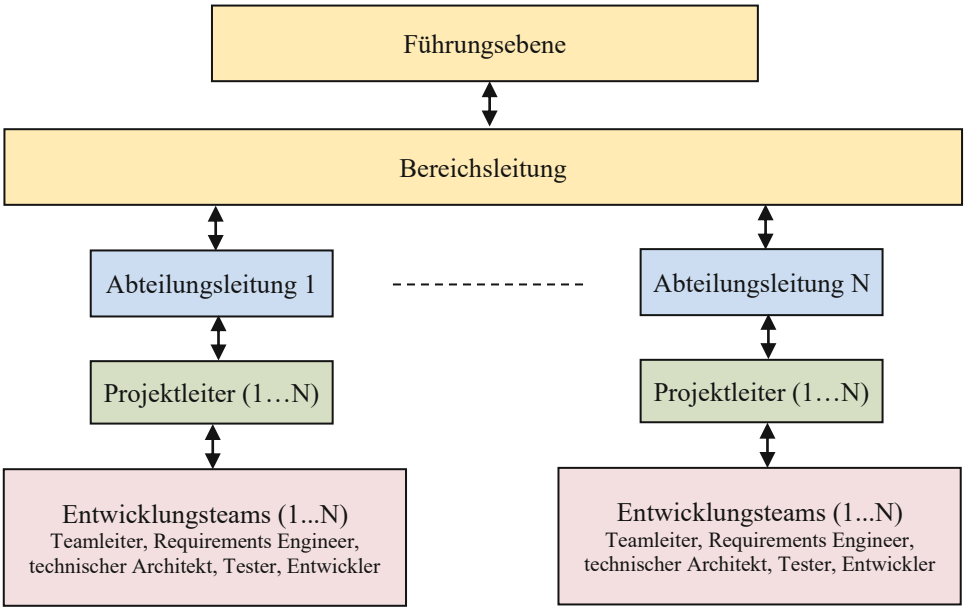


Abbildung 16: Projektorganisationsstruktur im Fallbeispiel 2

Der Product Owner gibt die Anforderungen an den Requirements Engineer weiter. Über dem Teamleiter gibt es dann einen Projektleiter, darüber die Abteilungsleitung. Über der Abteilungsleitung befindet sich die Bereichsleitung und an der Spitze der Hierarchie die Führungsebene.

Die Teamgröße auf der Entwicklungsebene liegt ca. bei 10-15 Personen. Die Projektmitarbeiteranzahl wird mit bis zu 170 Teilnehmern angegeben.

4.5.3 Vorgehensmodell und Projektphasen

Im Fallbeispiel 2 ist als Vorgehensmodell ist organisationsweit Scrum (vgl. Punkt 2.2.5.1) mit 2-wöchigen Sprints vorgesehen. Jeden Monat gibt es ein Release. Einige Teams haben jedoch die Freiheit auf Kanban (vgl. Punkt 2.2.5.4) umzusteigen, wenn sie ihr Modell ausformuliert haben und von der Abteilungsleitung abgesegnet bekommen. Der Umstieg auf Kanban wird von den Teammitgliedern damit argumentiert, da in gewissen Teilbereichen zu häufigen Änderungen der Prioritäten gibt und eine Fertigstellung zum geplanten Teilrelease nicht möglich ist. Ist nach vier Wochen ein Release fertig, kommt dieser Stand auf die Qualitätssicherungsumgebung. Dort werden Tests und Abnahmetests gemeinsam mit dem Kunden vorgenommen. Ist dieser Stand stabil, wird er auf der Produktionsumgebung ausgeliefert.

4.5.4 Zeitplan

Ein genauer Zeitraum wurde von den Interviewten nicht genannt. Das Projekt läuft jedoch bereits mehrere Jahre und ist zum aktuellen Stand 2025 noch nicht fertiggestellt.

4.5.5 Thematische Analyse der Interviews

Die thematische Analyse für Fallbeispiel 2 wird nach dem gleichen Vorgehen wie bei Fallbeispiel 1 durchgeführt (vgl. 4.4.7). Bereits definierte Codes und Themen aus Fallbeispiel 1 werden übernommen, sofern sie inhaltlich auf die neuen Interviewdaten zutreffen. Dabei wird eine einheitliche Benennung und Zuordnung verwendet, um die Vergleichbarkeit zwischen den Fallbeispielen zu gewährleisten.

Tabelle 10 zeigt die spezifischen Ergebnisse der Codierung und Themenbildung für Fallbeispiel 2 mit insgesamt 13 Themen und 71 Codes:

Anzahl der Interviews			5
Thema	**Code**	**%**	**# Dokumente**
Taskänderungen	fachlich über Requirements Engineer	20%	1
	Validierung und Einplanung nach Dringlichkeit	80%	4
	teamintern besprochen	20%	1
	Direkt wenn nicht zu kritisch (aufgrund von Bürokratie)	60%	3
Richtlinien, Grenzen, Normen oder Prinzipien	SonarQube	60%	3
	Renovate-Bot (Gitlab)	20%	1
	Checkstyle	20%	1
	Definition of Done	20%	1
	Zentrale Festlegung durch Spezialisten (1...n)	100%	5
	technische Architektur-Meetings	20%	1
	Organisationsweite Richtlinien	60%	3
	Doku (Wiki/Confluence/...)	20%	1
	zusätzliche Codequalitäten durch Team bzw. Teamleiter	80%	4
	Code Review	20%	1
Teamkultur	Teamkultur	20%	1
Meetings	Meetings bei Bedarf	40%	2
	Weekly (1-2 Wochen Zyklus)	60%	3
	Dailies	80%	4
	Teamleiter-Meetings (Test, Architektur, etc.)	80%	4
	Protokollierung	40%	2
Team Events & Belohnungen	Allgemeine Teamevents	40%	2
Komplexe Themen	4 Augen Prinzip & Reviews	40%	2
	Teamabsprache und Validierung	60%	3
Fortschrittsmessung	Task-Fortschritt (Monitoring) per Board	80%	4
	Im Daily	80%	4
	wöchentliche Meetings mit Leitung	20%	1
	Ticketing-Tool	20%	1

	Ticketstatus (%, Stunden oder Status)	80%	4
Auswahl und Zuteilung Tasks	Leitung an Team	60%	3
	Auswahl durch Teammitglieder aus Task-Liste	40%	2
	Zuteilung in Meeting demokratisch	20%	1
	Zuteilung nach Fachwissen/jenen der sich am besten auskennt	40%	2
	Team zu Team	20%	1
	priorisierte Taskliste	20%	1
	Requirements durch Product Owner	20%	1
Fortbildung	Teamwechsel	20%	1
	Zertifikate & Weiterbildung (aktiv)	20%	1
	Zeit für Verbesserungen	40%	2
	Themenwechsel	80%	4
	Rollenwechsel	60%	3
	Zeit Technologiewechsel & Updates	40%	2
	Udemy-Kurse	20%	1
Transparenz	Software & Ticketverwaltung	80%	4
	Teamübergreifende Meetings	20%	1
	Task-Fortschritt für jedes Mitglied	40%	2
	Status setzen	40%	2
	Direkte Kommunikation	40%	2
	Dailies (Transparenz)	100%	5
	Sprint-Preview	20%	1
	Boards (Kanban od. ähnliches)	80%	4
Wissensverteilung & Support	Code Review	80%	4
	Dokumentation	20%	1
	Übergabemeetings	40%	2
	Direkte Nachfrage (Sms, Telefon)	20%	1
	Taskrotation	60%	3
	Sprint-Review	40%	2
	Pair-Programming	60%	3
	Mehrere Personen pro Thema	60%	3
Reflexion & Lernprozess	Retrospektive & Review	100%	5
Aufwandsschätzung	Schätzung über mehrere Iterationen	100%	5
	Priorisierung von Tasks	40%	2
	Durch Teammitglieder	100%	5
	Aufteilung in Subtasks	20%	1
	projektspezifische Einheitenschätzung	40%	2
Kommunikation	Telefon	20%	1
	direkte Kommunikation	60%	3
	Ticket-Tool	20%	1
	Chat	100%	5
	Meetings	60%	3

	E-Mail	20%	1
	Wiki/Confluence/...	60%	3

Tabelle 10: Vorkommen der 71 Codes im Fallbeispiel 2

4.6 Single Case Study: Gesundheitsbereich (Fallbeispiel 3)

Fallbeispiel 3 widmet sich der Infrastruktur im Gesundheitsbereich eines Landes, zu dem sehr viele verschiedene Teilbereiche und Applikationen, sowohl im Software- als auch Hardwarebereich gehören. Das Produkt gliedert sich in mehrere Teilprojekte. Ein Teilbereich widmet sich der Hardware, welche als Anbindung an die Software benötigt wird. Die Teams designen nicht nur die Hardware, sondern auch das Betriebssystem und alles, was dazu gehört. Im Bereich der Software gibt es Teilprojekte für die mobile App-Entwicklung (Android und iOS), sowie nicht mobilen Anwendungen und Schnittstellen. Aufgrund der sensiblen Daten ist für dieses Projekt ein umfangreicher Zertifizierungsprozess notwendig.

4.6.1 Projektziel

Das Projektziel ist der Zugang und Austausch von Gesundheitsinformationen von versicherten Bürgern eines Landes. Ein elektronischer Akt eines Bürgers, welcher über Apps zugänglich ist und über verschiedene Krankenkassen zugänglich gemacht wird. Gesundheitsrelevante Daten können anschließend in diesem elektronischen Akt abgelegt werden. Damit sind alle wichtigen Informationen jederzeit digitalisiert abrufbar und müssen nicht über Umwege angefordert werden. Die Kontrolle dieser Daten obliegt allein dem Versicherten. Die Hardware, beinhaltet ein Fachmodul, dass den Zugang für die Kommunikation mit den Gesundheitsdaten bzw. der dazugehörigen Software zur Verfügung stellt.

4.6.2 Projektorganisationsstruktur, Rollen und Teams

Die Projektstruktur für das Fallbeispiel 3 ist aufgrund der vielen unterschiedlichen Aufgabenbereiche vor allem auf horizontaler Ebene sehr breit aufgestellt. Es gibt einen Projektsponsor, sowie die Geschäftsführung an oberster Stelle. Darunter gibt es in den Teilprojekten jeweils Projektleiter. Da die Projekte sehr groß sind, liegt die Verantwortung nicht allein an einem Projektleiter, sondern an mehreren. Es sind somit ganze Projektleitungs-Teams von bis zu zehn Leuten, welche die Projekte führen. Es gibt viele verschiedene Bereiche wie z.B. App-Entwicklung, Identität- und Authentifizierungslösungen, Backend, etc. In den Teilprojekten gibt es Teilprojektleiter, die entweder Teams führen, Kundenansprachen vornehmen oder Querschnittsaspekte erfüllen wie

Security, Zertifizierungen oder Test. Innerhalb der Teilprojekte werden die Teams von Team-Leads geführt, deren Mitarbeiter intern wieder bestimmten Rollen zugeteilt sind.

4.6.3 Vorgehensmodell und Projektphasen

Außen herum ist das Vorgehensmodell dieses Fallbeispiels ein Wasserfall-Modell (vgl. 2.2.5.3). Die Deadline des nächsten Releases ist mit 01.01 des nächsten Jahres vorgegeben. Ebenfalls vorgegeben ist der Zeitraum für die Überprüfung und das Security Gutachten. Das Produkt muss vor der Auslieferung von einer Zulassungsstelle freigegeben werden. Am Anfang der Entwicklung wird die Spezifikation übergeben, die jedoch überwiegend noch nicht vollständig fertig ausdefiniert ist. Während der kurzen Zeit der eigentlichen Entwicklungsphase werden kleinere Zwischenreleases mit Meilensteinen eingeführt. Innerhalb der Teilprojekte wird eine Kombination aus Scrum (vgl. 2.2.5.1) und Kanban (vgl. 2.2.5.4) verwendet. Am Anfang des Projekts wird meist Scrum verwendet, da hier ein sauberer Backlog abgearbeitet werden kann. Gegen Ende der Entwicklungsphase lässt sich der Scrum-Prozess jedoch nicht mehr komplett isoliert durchführen. Es muss nach Funktionalität und Priorität entwickelt werden, da sich die gesetzliche Deadline nicht verschieben lässt. Daher wird im letzten Abschnitt der Entwicklung meist auf Kanban umgestiegen.

4.6.4 Zeitplan

Die Releases sind typischerweise aufgrund gesetzlicher Anforderungen festgelegt und werden mit 01.01 eines Jahres produktiv gesetzt. In den letzten zwei Monaten vor dem Produktivdatum unterliegt das Release einer sicherheitstechnischen und funktionalen Prüfung. Die großen Features sind somit jährlich getaktet. Gelegentlich kommt es vor, dass es unter dem Jahr ein kleines Feature-Release gibt. Hotfix- und Bugfix-Releases sind quartalsmäßig vorgesehen. Die Entwicklung ist in mehrere Phasen bzw. Ausbaustufen gegliedert. Phase 1 war im Jahr 2022 bereits abgeschlossen. Im Jahr 2022 ist die Phase 2 umgesetzt worden und im Jahr 2023 ist die Phase 3 eingeplant. Die Phase 4 folgt dann im darauffolgenden Jahr, usw.

4.6.5 Thematische Analyse der Interviews

Das Vorgehen der thematischen Analyse ist äquivalent wie im Fallbeispiel 1 (vgl. 4.4.7) und Fallbeispiel 2 (vg. 4.5.5). Bereits definierte Codes und Themen aus den vorherigen Fallbeispielen werden übernommen, sofern sie inhaltlich auf die neuen Interviewdaten zutreffen. Tabelle 11 zeigt das Ergebnis mit insgesamt 14 Themen und 73 Codes:

Anzahl der Interviews			5
Thema	**Code**	**%**	**# Dokumente**
Taskänderungen	Prozess wird beschleunigt	20%	1
	Validierung und Einplanung nach Dringlichkeit	80%	4
	teamintern besprochen	20%	1
	Direkt wenn nicht zu kritisch (aufgrund von Bürokratie)	60%	3
	Absprachen mit Management	40%	2
Richtlinien, Grenzen, Normen oder Prinzipien	Definition of Done	20%	1
	Zulassungsverfahren (e.g. CC-Verfahren)	60%	3
	Zentrale Festlegung durch Spezialisten (1...n)	40%	2
	Organisationsweite Richtlinien	80%	4
	Doku (Wiki/Confluence/...)	60%	3
	Review nach Checkliste	40%	2
	zusätzliche Codequalitäten durch Team bzw. Teamleiter	60%	3
	Code Review	40%	2
Teamkultur	Teamkultur	80%	4
Meetings	Weekly (1-2 Wochen Zyklus)	60%	3
	Dailies	80%	4
	Teamleiter-Meetings (Test, Architektur, etc.)	20%	1
Team Events & Belohnungen	Sachgeschenke	20%	1
	Prämien	20%	1
	Allgemeine Teamevents	60%	3
Komplexe Themen	Ressourcen auslagern, Komplexität intern behalten	20%	1
	4 Augen Prinzip & Reviews	60%	3
	Teamabsprache und Validierung	100%	5
Fortschrittsmessung	Zwischenreleases	20%	1
	Task-Fortschritt (Monitoring) per Board	60%	3
	Im Daily	60%	3
	wöchentliche Meetings mit Leitung	40%	2
	Tracking über Epics (mehrere Stories)	20%	1
	Ticketing-Tool	60%	3
	Ticketstatus (%, Stunden oder Status)	100%	5
Auswahl und Zuteilung Tasks	Leitung an Team	80%	4
	Auswahl durch Teammitglieder aus Task-Liste	80%	4
	Zuteilung in Meeting demokratisch	20%	1
	Zuteilung nach Fachwissen/jenen der sich am besten auskennt	20%	1
	Team zu Team	20%	1
	priorisierte Taskliste	40%	2
Fortbildung	Zertifikate & Weiterbildung (aktiv)	60%	3
	Zeit für Verbesserungen	40%	2
	Themenwechsel	60%	3
	Rollenwechsel	60%	3
	Zeit Technologiewechsel & Updates	60%	3

Transparenz	Software & Ticketverwaltung	80%	4
	Task-Fortschritt für jedes Mitglied	40%	2
	Status setzen	60%	3
	Direkte Kommunikation	20%	1
	Dailies (Transparenz)	100%	5
	Boards (Kanban od. ähnliches)	20%	1
Wissensverteilung & Support	Code Review	100%	5
	JavaDoc (Code-Kommentare)	40%	2
	Dokumentation	80%	4
	Direkte Nachfrage (Sms, Telefon)	20%	1
	Taskrotation	80%	4
	Sprint-Review	20%	1
	Zuteilung an Andere bei Leerlauf	20%	1
	Pair-Programming	20%	1
	Mehrere Personen pro Thema	60%	3
	Liste mit Zuständigkeiten	20%	1
Reflexion & Lern-prozess	Retrospektive & Review	80%	4
	Mantras	20%	1
Aufwandsschätzung	Schätzung über mehrere Iterationen	80%	4
	Durch Teammitglieder	80%	4
	Aufteilung in Subtasks	20%	1
	projektspezifische Einheitenschätzung	20%	1
Kommunikation	Telefon	40%	2
	direkte Kommunikation	40%	2
	Ticket-Tool	20%	1
	Jira	100%	5
	Jitsi	60%	3
	Chat	80%	4
	Meetings	40%	2
	E-Mail	80%	4
	Videokonferenzen	20%	1
	Wiki/Confluence/...	80%	4

Tabelle 11: Vorkommen der 73 Codes im Fallbeispiel 3

5 Multiple Case Study

In diesem Kapitel wird die Multiple Case Study beschrieben. Die Ergebnisse der Single Case Studies aus Kapitel 4 werden zusammengeführt und auf Gemeinsamkeiten untersucht. In den folgenden Unterkapiteln wird auf die thematischen Überschneidungen der Codes eingegangen.

Für das finale Ergebnis werden ausschließlich jene Codes berücksichtigt, die in allen drei Fallbeispielen vorkommen. Diese Herangehensweise stellt sicher, dass nur die am stärksten übereinstimmenden Inhalte mit der höchsten Aussagekraft in die Analyse einfließen. Nur aus dieser gemeinsamen Schnittmenge werden konkrete Best Practices abgeleitet und inhaltlich ausgearbeitet. Codes, die lediglich in ein oder zwei Fallbeispielen vorkommen, werden zwar dokumentiert, aber nicht zu vollständigen Best Practices weiterentwickelt.

5.1 Auswertung der Multiple Case Study

Die Single Case Study (siehe Kapitel 4) zeigt die Auswertung der Themen und Codes für jedes einzelne Fallbeispiel. Die Auswertung der Multiple Case Study vereint die Themen mit ihren Codes über alle drei Fallbeispiele.

Da bereits bei der Single Case Study vorausschauend auf eine einheitliche Benennung der Themen und Codes geachtet wird (siehe Punkt 4.4.7 ff.), findet nun eine Auswertung der Überschneidung der Codes in zwei, bzw. in allen drei Fallbeispielen statt. Insgesamt werden folgende Überschneidungen in der Multiple Case Study beachtet:

I. Themen & Codes in den Fallbeispielen 1, 2 und 3 (siehe Punkt 5.2.1)
II. Themen & Codes in den Fallbeispielen 1 und 2
    i. inklusive Überschneidungen mit Fallbeispiel 3 (siehe Punkt 5.2.2)
    ii. exklusive Überschneidungen mit Fallbeispiel 3 (siehe Punkt 5.2.5.1)
III. Themen & Codes in den Fallbeispielen 1 und 3
    i. inklusive Überschneidungen mit Fallbeispiel 2 (siehe Punkt 5.2.3)
    ii. exklusive Überschneidungen mit Fallbeispiel 2 (siehe Punkt 5.2.5.2)
IV. Themen & Codes in den Fallbeispielen 2 und 3
    i. inklusive Überschneidungen mit Fallbeispiel 1 (siehe Punkt 5.2.4)
    ii. exklusive Überschneidungen mit Fallbeispiel 1 (siehe Punkt 5.2.5.3)

Für die Analyse der statistischen Verteilung sowie der Überschneidungen von Codes und Themen in den drei Fallbeispielen wird ein eigens entwickeltes Java-Programm verwendet. Dieses verarbeitet die aus MAXQDA exportierten Daten automatisiert und ermöglicht eine strukturierte Auswertung der thematischen Gemeinsamkeiten und Unterschiede.

5.2 Die Code Themen

Die Tabelle 12 zeigt das gesamte Codesystem mit insgesamt 813 Codes über 17 Code-Hauptthemen. Die Anzahl der vergebenen Codes pro Dokument ist in der Abbildung 17 in der rechten Spalte ersichtlich. Die 17 Hauptthemen umfassen wiederum die dazugehörigen Codes und Subcodes.

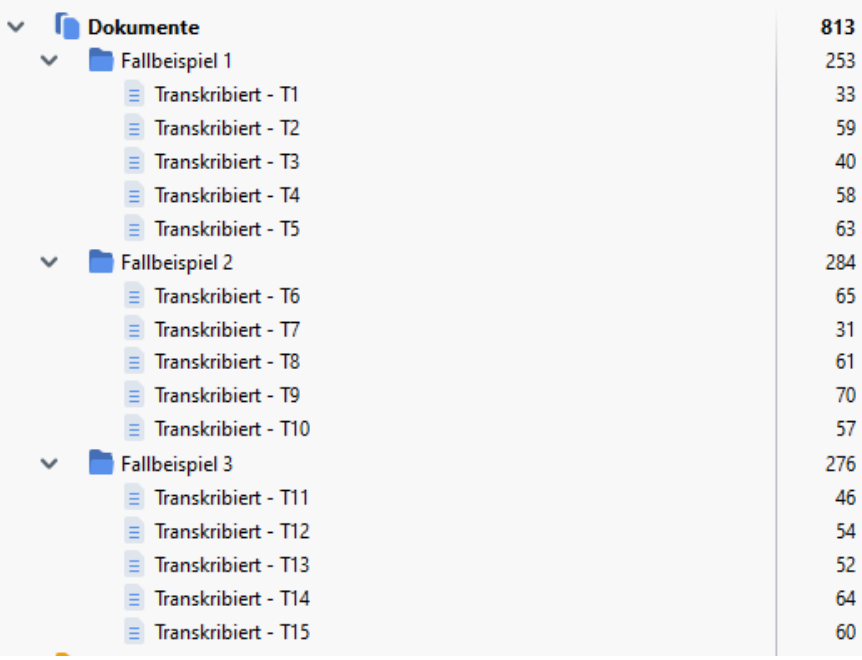


Abbildung 17: Anzahl vergebener Codes pro Dokument

Codesystem - Themen	Anzahl Gesamt
Teamkultur	6
Metadaten	92
Negativ-Codes	15
Umgebung	4
Aufwandsschätzung	62
Komplexe Themen	44
Team Events & Belohnungen	13
Reflexion & Lernprozess	21
Wissensverteilung & Support	86

Kommunikation	90
Fortbildung	43
Meetings	49
Fortschrittsmessung	62
Transparenz	93
Auswahl und Zuteilung Tasks	41
Taskänderungen	26
Richtlinien, Grenzen, Normen & Prinzipien	66
Summe	**813**

Tabelle 12: Übersicht der Code-Themen

In den folgenden Kapiteln werden die Überschneidungen zwischen den Fallbeispielen dargestellt. Die zwei Themen Negativ-Codes und Metadaten werden nicht in die Auswertung übernommen, da die Metadaten rein zur Zuordnung erstellt wurden, und negative Codes nicht Teil der Best Practices sind. Die Abbildung 18 illustriert die Schnittmengen der Codes.

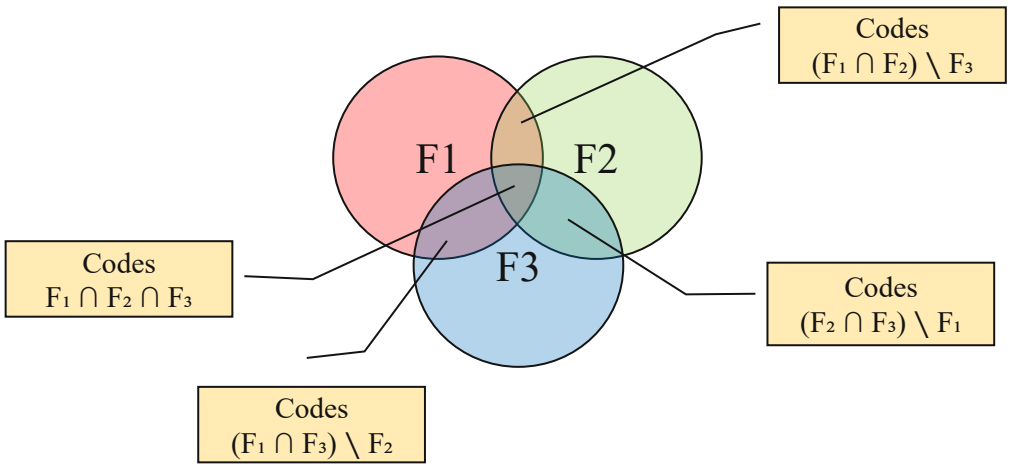


Abbildung 18: Schnittmengen der Codes

5.2.1 Überschneidungen Fallbeispiel 1, 2 und Fallbeispiel 3

Die folgende Tabelle 13 listet alle Themen und Codes mit ihren Vorkommen, welche in allen drei Fallbeispielen vorkommen ($F_1 \cap F_2 \cap F_3$). Die Spalte Vorkommen in Prozent wird in der Tabelle gerundet dargestellt:

Anzahl der Dokumente				15
Nr.	**Thema**	**Code**	**%-Vor-kommen**	**Anzahl Do-kumente**
1	Transparenz	Software & Ticketverwaltung	80%	12
		Task-Fortschritt für jedes Mitglied	47%	7
		Status setzen	60%	9
		Direkte Kommunikation	27%	4
		Dailies (Transparenz)	87%	13
		Boards (Kanban od. ähnliches)	67%	10
2	Kommunikation	Telefon	40%	6
		direkte Kommunikation	67%	10
		Ticket-Tool	27%	4
		Chat	87%	13
		Meetings	47%	7
		E-Mail	67%	10
		Wiki/Confluence/...	53%	8
3	Reflexion & Lern-prozess	Retrospektive & Review	67%	10
4	Wissensverteilung & Support	Code Review	93%	14
		Dokumentation	40%	6
		Taskrotation	53%	8
		Pair-Programming	40%	6
		Direkte Nachfrage (Sms, Telefon)	20%	3
		Mehrere Personen pro Thema	60%	9
5	Fortbildung	Zertifikate & Weiterbildung (aktiv)	33%	5
		Zeit für Verbesserungen	33%	5
		Themenwechsel	67%	10
		Rollenwechsel	60%	9
		Zeit Technologiewechsel & Updates	40%	6
6	Richtlinien, Gren-zen, Normen oder Prinzipien	Doku (Wiki/Confluence/...)	53%	8
		Organisationsweite Richtlinien	73%	11
		Festlegung zentral durch Spezialisten (1...n)	73%	11
		zusätzliche Codequalitäten durch Team bzw. Teamleiter	60%	9
7	Meetings	Weekly (1-2 Wochen Zyklus)	60%	9
		Dailies	80%	12
		Teamleiter-Meetings (Test, Architektur, etc.)	53%	8
8	Fortschrittsmessung	wöchentliche Meetings mit Leitung	27%	4
		Task-Fortschritt (Monitoring) per Board	73%	11
		Im Daily	53%	8
		Ticketstatus (%, Stunden oder Status)	87%	13
9	Auswahl und Zutei-lung Tasks	Leitung an Team	80%	12
		Zuteilung nach Fachwissen/jenen der sich am besten auskennt	40%	6
		Auswahl durch Teammitglieder aus Task-Liste	73%	11

10	Taskänderungen	Validierung und Einplanung nach Dringlichkeit	60%	9
		Direkt wenn nicht zu kritisch (aufgrund von Bürokratie)	67%	10
11	Team Events & Belohnungen	Allgemeine Teamevents	60%	9
12	Komplexe Themen	4 Augen Prinzip & Reviews	67%	10
		Teamabsprache und Validierung	80%	12
13	Aufwandsschätzung	Schätzung über mehrere Iterationen	67%	10
		Durch Teammitglieder	93%	14
		Aufteilung in Subtasks	20%	3
		projektspezifische Einheitenschätzung	33%	5

Tabelle 13: Die 48 Codes in 13 Themen in den Fallbeispielen 1, 2 und 3

5.2.2 Überschneidungen Fallbeispiel 1 und Fallbeispiel 2

Die folgende Tabelle 14 listet alle Themen und Codes mit ihren Vorkommen, welche in den beiden Fallbeispielen vorkommen ($F_1 \cap F_2$):

Anzahl der Dokumente				10
Nr.	**Thema**	**Code**	**%-Vorkommen**	**Anzahl Dokumente**
1	Taskänderungen	Validierung und Einplanung nach Dringlichkeit	50%	5
		Direkt wenn nicht zu kritisch (aufgrund von Bürokratie)	70%	7
2	Richtlinien, Grenzen, Normen oder Prinzipien	SonarQube	50%	5
		Organisationsweite Richtlinien	70%	7
		Doku (Wiki/Confluence/...)	50%	5
		Checkstyle	30%	3
		Festlegung zentral durch Spezialisten (1...n)	90%	9
		zusätzliche Codequalitäten durch Team bzw. Teamleiter	60%	6
3	Meetings	Meetings bei Bedarf	40%	4
		Weekly (1-2 Wochen Zyklus)	60%	6
		Dailies	80%	8
		Teamleiter-Meetings (Test, Architektur, etc.)	70%	7
4	Team Events & Belohnungen	Allgemeine Teamevents	60%	6
5	Komplexe Themen	4 Augen Prinzip & Reviews	70%	7
		Teamabsprache und Validierung	70%	7
6	Fortschrittsmessung	Task-Fortschritt (Monitoring) per Board	80%	8
		Im Daily	50%	5
		wöchentliche Meetings mit Leitung	20%	2

		Ticketstatus (%, Stunden oder Status)	80%	8
7	Auswahl und Zuteilung Tasks	Leitung an Team	80%	8
		Zuteilung nach Fachwissen/jenen der sich am besten auskennt	50%	5
		Auswahl durch Teammitglieder aus Task-Liste	70%	7
8	Fortbildung	Teamwechsel	30%	3
		Zertifikate & Weiterbildung (aktiv)	20%	2
		Zeit für Verbesserungen	30%	3
		Themenwechsel	70%	7
		Rollenwechsel	60%	6
		Zeit Technologiewechsel & Updates	30%	3
9	Transparenz	Software & Ticketverwaltung	80%	8
		Task-Fortschritt für jedes Mitglied	50%	5
		Status setzen	50%	5
		Direkte Kommunikation	30%	3
		Dailies (Transparenz)	80%	8
		Boards (Kanban od. ähnliches)	90%	9
10	Wissensverteilung & Support	Code Review	90%	9
		Dokumentation	20%	2
		Taskrotation	40%	4
		Pair-Programming	50%	5
		Übergabemeetings	40%	4
		Direkte Nachfrage (Sms, Telefon)	20%	2
		Mehrere Personen pro Thema	70%	7
11	Reflexion & Lernprozess	Retrospektive & Review	60%	6
12	Aufwandsschätzung	Schätzung über mehrere Iterationen	60%	6
		Durch Teammitglieder	100%	10
		Aufteilung in Subtasks	20%	2
		projektspezifische Einheitenschätzung	40%	4
13	Kommunikation	direkte Kommunikation	80%	8
		Ticket-Tool	30%	3
		Chat	90%	9
		Meetings	50%	5
		Telefon	40%	4
		E-Mail	60%	6
		Wiki/Confluence/...	40%	4

Tabelle 14: Die 53 Codes in 13 Themen in den Fallbeispielen 1 und 2

5.2.3 Überschneidungen Fallbeispiel 1 und Fallbeispiel 3

Die folgende Tabelle 15 listet alle Themen und Codes mit ihren Vorkommen, welche in den beiden Fallbeispielen vorkommen ($F_1 \cap F_3$):

Anzahl der Dokumente				10
Nr.	**Thema**	**Code**	**%-Vorkommen**	**Anzahl Dokumente**
1	Taskänderungen	Validierung und Einplanung nach Dringlichkeit	50%	5
		Direkt wenn nicht zu kritisch (aufgrund von Bürokratie)	70%	7
2	Richtlinien, Grenzen, Normen oder Prinzipien	Doku (Wiki/Confluence/...)	70%	7
		Organisationsweite Richtlinien	80%	8
		Festlegung zentral durch Spezialisten (1...n)	60%	6
		Review nach Checkliste	40%	4
		zusätzliche Codequalitäten durch Team bzw. Teamleiter	50%	5
3	Meetings	Weekly (1-2 Wochen Zyklus)	60%	6
		Dailies	80%	8
		Teamleiter-Meetings (Test, Architektur, etc.)	40%	4
4	Team Events & Belohnungen	Allgemeine Teamevents	70%	7
5	Komplexe Themen	4 Augen Prinzip & Reviews	80%	8
		Teamabsprache und Validierung	90%	9
6	Fortschrittsmessung	Im Daily	40%	4
		Task-Fortschritt (Monitoring) per Board	70%	7
		wöchentliche Meetings mit Leitung	30%	3
		Ticketstatus (%, Stunden oder Status)	90%	9
7	Auswahl und Zuteilung Tasks	Leitung an Team	90%	9
		Zuteilung nach Fachwissen/jenen der sich am besten auskennt	40%	4
		Auswahl durch Teammitglieder aus Task-Liste	90%	9
8	Fortbildung	Zertifikate & Weiterbildung (aktiv)	40%	4
		Zeit für Verbesserungen	30%	3
		Themenwechsel	60%	6
		Rollenwechsel	60%	6
		Zeit Technologiewechsel & Updates	40%	4
9	Transparenz	Software & Ticketverwaltung	80%	8
		Task-Fortschritt für jedes Mitglied	50%	5
		Status setzen	60%	6
		Direkte Kommunikation	20%	2

		Dailies (Transparenz)	80%	8
		Boards (Kanban od. ähnliches)	60%	6
10	Wissensverteilung & Support	Zuteilung an Andere bei Leerlauf	20%	2
		Code Review	100%	10
		JavaDoc (Code-Kommentare)	30%	3
		Dokumentation	50%	5
		Taskrotation	50%	5
		Pair-Programming	30%	3
		Direkte Nachfrage (Sms, Telefon)	20%	2
		Mehrere Personen pro Thema	60%	6
11	Reflexion & Lernprozess	Retrospektive & Review	50%	5
12	Aufwandsschätzung	Schätzung über mehrere Iterationen	50%	5
		Durch Teammitglieder	90%	9
		Aufteilung in Subtasks	20%	2
		projektspezifische Einheitenschätzung	30%	3
13	Kommunikation	Telefon	50%	5
		direkte Kommunikation	70%	7
		Ticket-Tool	30%	3
		Chat	80%	8
		Meetings	40%	4
		E-Mail	90%	9
		Wiki/Confluence/...	50%	5

Tabelle 15: Die 51 Codes in 13 Themen in den Fallbeispielen 1 und 3

5.2.4 Überschneidungen Fallbeispiel 2 und Fallbeispiel 3

Die folgende Tabelle 16 listet alle Themen und Codes mit ihren Vorkommen, welche in den beiden Fallbeispielen vorkommen ($F_2 \cap F_3$):

Anzahl der Dokumente				10
Nr.	**Thema**	**Code**	**%-Vorkommen**	**Anzahl Dokumente**
1	Taskänderungen	Validierung und Einplanung nach Dringlichkeit	80%	8
		teamintern besprochen	20%	2
		Direkt wenn nicht zu kritisch (aufgrund von Bürokratie)	60%	6
2	Richtlinien, Grenzen, Normen oder Prinzipien	Organisationsweite Richtlinien	70%	7
		Doku (Wiki/Confluence/...)	40%	4
		Definition of Done	20%	2
		Festlegung zentral durch Spezialisten (1...n)	70%	7

		zusätzliche Codequalitäten durch Team bzw. Teamleiter	70%	7
		Code Review	30%	3
3	Teamkultur	Teamkultur	50%	5
4	Meetings	Weekly (1-2 Wochen Zyklus)	60%	6
		Dailies	80%	8
		Teamleiter-Meetings (Test, Architektur, etc.)	50%	5
	Team Events & Belohnungen	Allgemeine Teamevents	50%	5
5	Komplexe Themen	4 Augen Prinzip & Reviews	50%	5
		Teamabsprache und Validierung	80%	8
6	Fortschrittsmessung	Im Daily	70%	7
		Task-Fortschritt (Monitoring) per Board	70%	7
		wöchentliche Meetings mit Leitung	30%	3
		Ticketing-Tool	40%	4
		Ticketstatus (%, Stunden oder Status)	90%	9
7	Auswahl und Zuteilung Tasks	Leitung an Team	70%	7
		Auswahl durch Teammitglieder aus Task-Liste	60%	6
		Zuteilung in Meeting demokratisch	20%	2
		Zuteilung nach Fachwissen/jenen der sich am besten auskennt	30%	3
		Team zu Team	20%	2
		priorisierte Taskliste	30%	3
8	Fortbildung	Zertifikate & Weiterbildung (aktiv)	40%	4
		Zeit für Verbesserungen	40%	4
		Themenwechsel	70%	7
		Rollenwechsel	60%	6
		Zeit Technologiewechsel & Updates	50%	5
9	Transparenz	Software & Ticketverwaltung	80%	8
		Direkte Kommunikation	30%	3
		Task-Fortschritt für jedes Mitglied	40%	4
		Status setzen	60%	6
		Dailies (Transparenz)	100%	10
		Boards (Kanban od. ähnliches)	50%	5
10	Wissensverteilung & Support	Code Review	90%	9
		Dokumentation	50%	5
		Pair-Programming	40%	4
		Direkte Nachfrage (Sms, Telefon)	20%	2
		Taskrotation	70%	7
		Sprint-Review	30%	3

		Mehrere Personen pro Thema	60%	6
	Reflexion & Lern-prozess	Retrospektive & Review	90%	9
11	Aufwandsschätzung	Schätzung über mehrere Iteratio-nen	90%	9
		Durch Teammitglieder	90%	9
		Aufteilung in Subtasks	20%	2
		projektspezifische Einheiten-schätzung	30%	3
12	Kommunikation	Telefon	30%	3
		direkte Kommunikation	50%	5
		Ticket-Tool	20%	2
		Chat	90%	9
		Meetings	50%	5
		E-Mail	50%	5
		Wiki/Confluence/...	70%	7

Tabelle 16: Die 57 Codes in 12 Themen in den Fallbeispielen 2 und 3

5.2.5 Codes die nur in zwei Fallbeispielen gemeinsam existieren

Die folgenden Tabellen zeigen jene Codes, die ausschließlich in der Schnittmenge zwischen zwei Fallbeispielen vorkommen. Es werden also alle Codes zwischen den jeweiligen Fallbeispielen ermittelt und jene entfernt, die in allen drei Fallbeispielen gemeinsam vorkommen. Dabei ist zu beachten, dass die Codes inhaltlich auf unterschiedlichen Abstraktionsebenen angesiedelt sind – von allgemeinen Methoden bis hin zu konkreten Tools.

Ein Beispiel hierfür sind Begriffe wie Checkstyle oder SonarQube, die einen sehr spezifischen Toolbezug aufweisen. Auch wenn solche Tools nicht in allen drei Fallbeispielen namentlich genannt wurden und daher nicht als gemeinsame Codes gewertet werden, kann die ihnen zugrunde liegende übergreifende Codierung (z. B. „zusätzliche Codequalitäten durch Team bzw. Teamleiter“) sehr wohl übergreifend abgedeckt sein. In solchen Fällen werden entsprechende Tools in den Beschreibungen der Best Practices exemplarisch erwähnt, ohne dass sie als Voraussetzung für die Anwendung gelten. Die Best Practices selbst sind grundsätzlich hersteller- und plattformunabhängig formuliert.

5.2.5.1 Gemeinsame Codes im Fallbeispiel 1 und im Fallbeispiel 2, die nicht im Fallbeispiel 3 auftreten

Die folgende Tabelle 17 listet alle Codes, die ausschließlich in der Schnittmenge von Fallbeispiel 1 und Fallbeispiel 2 vorkommen, jedoch nicht im Fallbeispiel 3 enthalten sind ($(F_1 \cap F_2) \setminus F_3$):

Thema	Code	Anmerkung
Fortbildung	Teamwechsel	Im Fallbeispiel 3 wird kein direkter Verweis auf Teamwechsel als Fortbildungsmaßnahme gemacht. Stattdessen liegt der Fokus auf Rollen- und Themenwechsel.
Meetings	Meetings bei Bedarf	Im Fallbeispiel 3 gibt es keine spezifischen Hinweise auf flexible Meetings. Der Fokus liegt eher auf regelmäßigen und strukturierten Meetings wie Dailies.
Richtlinien, Grenzen, Normen oder Prinzipien	Checkstyle	Checkstyle wird im Fallbeispiel 3 nicht erwähnt. Stattdessen liegt dort der Fokus auf allgemeiner Codequalität und Richtlinien, ohne spezielle Tools zu nennen.
	SonarQube	Ähnlich wie bei Checkstyle wird SonarQube im Fallbeispiel 3 nicht erwähnt, obwohl allgemeine Qualitätskontrollen angesprochen werden.
Wissensverteilung & Support	Übergabemeetings	Übergabemeetings werden im Fallbeispiel 3 nicht explizit erwähnt. Stattdessen wird dort der Wissensaustausch durch Pair-Programming und Dokumentation hervorgehoben.

Tabelle 17: Codes $(F_1 \cap F_2) \setminus F_3$

5.2.5.2 Gemeinsame Codes im Fallbeispiel 1 und im Fallbeispiel 3, die nicht im Fallbeispiel 2 auftreten

Die folgende Tabelle 18 listet alle Codes, die ausschließlich in der Schnittmenge von Fallbeispiel 1 und Fallbeispiel 3 vorkommen, jedoch nicht im Fallbeispiel 2 enthalten sind ($(F_1 \cap F_3) \setminus F_2$):

Thema	Code	Anmerkung
Wissensverteilung & Support	Zuteilung an Andere bei Leerlauf	Im Fallbeispiel 2 wurde keine explizite Erwähnung von Aufgabenübergaben bei Leerlauf gefunden. Stattdessen liegt der Fokus auf anderen Zuteilungsmechanismen.
	JavaDoc (Code-Kommentare)	JavaDoc wird im Fallbeispiel 2 nicht ausdrücklich erwähnt. Es könnte sein, dass es genutzt wird, aber nicht als Schwerpunkt der Dokumentation hervorgehoben wurde.
Richtlinien, Grenzen, Normen oder Prinzipien	Review nach Checkliste	Im Fallbeispiel 2 wurde der Review-Prozess mit einer Definition of Done beschrieben, jedoch ohne spezifische Erwähnung von Checklisten als Strukturierungsinstrument.

Tabelle 18: Codes $(F_1 \cap F_3) \setminus F_2$

5.2.5.3 Gemeinsame Codes im Fallbeispiel 2 und im Fallbeispiel 3, die nicht im Fallbeispiel 1 auftreten

Die folgende Tabelle 19 listet alle Codes, die ausschließlich in der Schnittmenge von Fallbeispiel 2 und Fallbeispiel 3 vorkommen, jedoch nicht im Fallbeispiel 1 enthalten sind ($(F_2 \cap F_3) \setminus F_1$):

Thema	Code	Anmerkung
Auswahl und Zuteilung Tasks	Zuteilung in Meeting demokratisch	Demokratische Zuteilung wird im Fallbeispiel 1 nicht beschrieben, wo die Zuteilung meist zentral oder selbstverantwortlich erfolgt.
	Team zu Team	Dieser Aspekt der Zusammenarbeit zwischen Teams wird im Fallbeispiel 1 nicht angesprochen.
	priorisierte Taskliste	Priorisierung von Aufgabenlisten wird im Fallbeispiel 1 nicht hervorgehoben.
Fortschrittsmessung	Ticketing-Tool	Im Fallbeispiel 1 wird die Fortschrittsmessung mit LFBs (Leistungsfortschrittsberichten) gemessen. LFBs sind Excel-Listen.
Richtlinien, Grenzen, Normen oder Prinzipien	Definition of Done	Die Definition of Done wird im Fallbeispiel 1 nicht ausdrücklich erwähnt, aber es gibt Checklisten.
	Code Review	Obwohl Code Reviews im Fallbeispiel 1 erwähnt werden, fehlt der spezifische Fokus auf deren systematische Durchführung.
Taskänderungen	teamintern besprochen	Dieser Aspekt wird im Fallbeispiel 1 nicht hervorgehoben, wo Änderungen eher zentral koordiniert werden.
Teamkultur	Teamkultur	Die Entwicklung einer expliziten Teamkultur wird im Fallbeispiel 1 nicht thematisiert.
Wissensverteilung & Support	Sprint-Review	Sprint-Reviews werden im Fallbeispiel 1 nicht als zentraler Bestandteil der Wissensverteilung hervorgehoben.

Tabelle 19: Codes $(F_2 \cap F_3) \setminus F_1$

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Multiple Case Study vorgestellt, die in Kapitel 5 beschrieben wird. Ziel der Untersuchung ist es, skalierbare Best Practices für autonome Teams in Softwaregroßprojekten zu identifizieren und diese mit dem bestehenden Modell der Balancing Acts von Hoda et al. zu vergleichen. Die aussagekräftigsten Ergebnisse basieren auf den Themen und Codes, die in allen drei Fallbeispielen übereinstimmen. Tabelle 13 fasst die identifizierten Themen und Best Practices zusammen, die in diesem Kapitel ausführlich dargestellt werden.

Zunächst werden in Kapitel 6.1 die Best Practices beschrieben, die aus den Fallstudien abgeleitet wurden (RQ1a). Darauf aufbauend folgt in Kapitel 6.2 die Analyse ihrer Skalierbarkeit im Abgleich mit dem Modell von Hoda et al. (RQ1b). In Kapitel 6.3 werden die Praktiken anschließend thematisch in 13 Kategorien eingeordnet (RQ2), bevor in Kapitel 6.4 die Kriterien dargelegt werden, anhand derer die Zuordnung der Practices zu diesen Kategorien erfolgt (RQ3).

Kapitel 6.5 bietet einen erneuten Vergleich der identifizierten Best Practices mit aktuellen wissenschaftlichen Erkenntnissen und Modellen aus der Literatur nach Fertigstellung der Ergebnisse (State of the Art aus dem Jahr 2024 und 2025) und Kapitel 6.6 beschreibt die Limitationen der Ergebnisse und gibt Hinweise auf mögliche Einflüsse und Einschränkungen der durchgeführten Analyse.

6.1 Benennung der Best Practices aus den Codes der Hauptthemen

Da die ursprünglichen Codes in Tabelle (siehe Tabelle 13) stichwortartig formuliert sind, werden diese in aussagekräftige Best Practices zusammengefasst, umbenannt und mit kurzen Beschreibungen sowie Beispielen aus den Fallbeispielen der Praxis versehen. Diese Umbenennung soll die praktische Bedeutung und bessere Beschreibung der Praktiken verdeutlichen. Die Ableitung der Best Practices erfolgt auf Basis der codierten Interviewstellen. Dabei werden alle Textstellen, die einem bestimmten Code zugeordnet sind, systematisch analysiert und deren Inhalte zu zusammenhängenden Aussagen verdichtet. Die Rückverfolgbarkeit funktioniert über die verwendete Software MAXQDA und gewährleistet, dass jede Best Practice durch konkrete Interviewaussagen belegt ist.

Für jedes der 13 Hauptthemen (siehe Punkt 6.3) werden spezifische Best Practices identifiziert, die aus den 48 Codes und deren zugehörigen Interviewphrasen abgeleitet sind. Diese bieten eine detaillierte Perspektive auf die Umsetzung der Praktiken innerhalb der Kategorien. Es ergeben sich insgesamt 23 ausformulierte Best Practices, wobei einige in mehreren Kategorien zugeteilt werden. Die Mehrfachzuteilung ergibt sich daraus, dass einzelne Interviewaussagen inhaltlich mehrere Aspekte berühren und deshalb auch mehreren Codes zugewiesen werden – selbst wenn diese unterschiedlichen Kategorien angehören. Ein Zitat kann zum Beispiel gleichzeitig auf „Wissensverteilung & Support“ und „Reflexion & Lernprozess“ hinweisen. Solche inhaltlichen Überschneidungen werden in der Analyse durch eine gemeinsame Best Practice zusammengefasst. In diesen Fällen wird jeweils auf die Ersterwähnung und ausführliche Beschreibung der entsprechenden Best Practice verwiesen, um Redundanzen zu vermeiden.

Die Titel der Best Practices werden somit bewusst allgemein gewählt, da einzelne Praktiken mehrere Themenbereiche gleichzeitig abdecken. Ziel ist es, inhaltliche Überschneidungen konsistent unter einem übergeordneten Namen zusammenzufassen und Wiederholungen in der Beschreibung zu vermeiden.

Am Ende des Am Ende dieses Kapitels zeigt die Tabelle 24 eine Gesamtübersicht über alle Best Practices mit ihren zugehörigen Kategorien und einer Kurzbeschreibung der kategorienspezifischen Umsetzung. Die in diesem Kapitel beschriebenen Best Practices stellen die Antwort auf die Forschungsfrage RQ1a dar: „Welche Praktiken lassen sich in Softwaregroßprojekten zur Unterstützung autonomer Teams identifizieren?“ und bilden zugleich die Grundlage für die Beantwortung der Forschungsfrage RQ1b in 6.2.

In einigen Best-Practice-Beschreibungen werden konkrete Tools wie Jira oder SonarQube beispielhaft erwähnt, um die praktische Umsetzung zu veranschaulichen (vgl. Kapitel 5.2.5). Diese Nennungen dienen ausschließlich der Illustration typischer Anwendungsszenarien. Die Best Practices selbst sind bewusst hersteller- und plattformunabhängig formuliert und orientieren sich an übergreifenden Methoden und Prinzipien.

6.1.1 Transparenz

Die Codes „Software & Ticketverwaltung“, „Task-Fortschritt für jedes Mitglied“, „Status setzen“, „Direkte Kommunikation“, „Dailies (Transparenz)“ und „Boards (Kanban oder ähnliches)“ werden zu folgenden Best Practices ausformuliert:

6.1.1.1 Toolunterstützter Workflow

Der Einsatz von Software-Tools spielt eine zentrale Rolle in agilen Teams, um Aufgaben zu verwalten, den Fortschritt zu verfolgen und Wissen zu sichern. Die eingesetzten Systeme decken dabei mehrere zentrale Bereiche gleichzeitig ab: Transparenz, Kommunikation, Wissensverteilung & Support sowie Fortschrittsmessung. Informationen werden digital zentral gespeichert, Arbeitsprozesse dokumentiert und teamübergreifend koordiniert. Der gezielte, integrierte Einsatz dieser digitalen Werkzeuge unterstützt agile Teams dabei, ihre täglichen Arbeitsprozesse transparent und effizient zu gestalten. Insbesondere in Daily Meetings nutzen die Teams die Tools aktiv, um den aktuellen Arbeitsstand sichtbar zu machen, Blocker zu erkennen und die nächsten Schritte abzustimmen. Die Toolnutzung ist dadurch nicht rein unterstützend, sondern integraler Bestandteil der täglichen Teamkoordination. Insbesondere in verteilten Projekten dienen die Tools als zentrale Schnittstellen zur Koordination, Dokumentation und Wissenssicherung.

Für die transparente Ticketverwaltung und Statusverfolgung setzen die Teams unterschiedliche Tools ein. Im Fallbeispiel 1 erfolgt die Aufgabenverwaltung hauptsächlich über GitLab-Boards, die als digitale Kanban-Systeme genutzt werden. Dort werden Aufgaben in verschiedene Status eingeteilt, typischerweise „Backlog“, „In Bearbeitung“, „In Review“ und „Abgeschlossen“, um jederzeit den Bearbeitungsstand nachvollziehen zu können. Kleinere Teams setzen zusätzlich auf physische Whiteboards, um ihre Workflows sichtbar zu machen. HPQC („HP Quality Center“) dient im Fallbeispiel 1 zudem als übergreifendes Requirements- und Defect-Management-Tool, in dem Aufgaben kategorisiert und mit Status versehen werden. In den Fallbeispielen 2 und 3 kommt dagegen Jira als zentrales Ticket- und Sprint-Management-Tool zum Einsatz. Die Teams nutzen dort ebenfalls Kanban- und Sprint-Boards mit standardisierten Statuskategorien. In einigen Teams von Fallbeispiel 2 wird ergänzend noch GitLab verwendet, wobei langfristig eine vollständige Umstellung auf Jira geplant ist. Zur Visualisierung der Verantwortlichkeiten werden häufig Avatare oder Initialen eingesetzt, während Labels zusätzliche Informationen wie Priorität, Blocker oder Themenzugehörigkeit markieren.

> *„Wir verwenden jetzt teamintern Großteils Gitlab für die Boards und ansonsten erweitert HPQC Requirements und Defects.“* – T3, Teamleiter

> *„Wir haben eigentlich immer Jira verwendet. Wir waren quasi die Vorreiter, wir haben auch immer Gitlab verwendet. Und es wird jetzt quasi umgestellt. Es sollen - alle eigentlich Jira und Gitlab werden, wenn es geht. Aber es ist stark gesplittet, einfach historisch bedingt haben die anderen Teams z.B. das RTC von IBM, das Rational Team Concert im Einsatz und haben Gerrit und manche Bit Bucket. Und ja,*

> *manche haben Gitlab. Sehr, sehr unterschiedliche Dinge. Aber es soll quasi alles zusammengeführt werden.“* – T6, Technischer Architekt

Neben der Verwaltung von Aufgaben spielen auch die Dokumentation und Wissenssicherung eine zentrale Rolle. Im Fallbeispiel 1 werden relevante Informationen in HPQC und internen Wikis gespeichert, darunter technische Dokumentationen, Architekturübersichten und Meeting-Protokolle. Im Fallbeispiel 2 und 3 hingegen nutzen die Teams Confluence, um Sprint-Reviews, Best Practices und technische Anleitungen zu dokumentieren. Besonders im Fallbeispiel 3 werden Jira und Confluence kombiniert genutzt, um API-Abstimmungen, How-To-Guides und Release-Übersichten zentral bereitzustellen. Die enge Verzahnung dieser Tools mit der Ticketverwaltung erleichtert es, Änderungen und Entscheidungen schnell nachzuvollziehen.

Auch die Fortschrittsmessung erfolgt über diese Systeme. Im Fallbeispiel 1 nutzen einige Teams GitLab-Dashboards, während andere auf manuelle Leistungsfortschrittsberichte zurückgreifen. Im Fallbeispiel 2 werden die Zeiterfassung und Aufwandsschätzungen direkt in Jira pro Ticket hinterlegt, sodass der Arbeitsaufwand messbar bleibt. Im Fallbeispiel 3 werden Sprint-Fortschritte regelmäßig in Jira und Confluence dokumentiert, um eine durchgehende Nachverfolgbarkeit zu gewährleisten.

Durch die Kombination von Ticket-Systemen, Dokumentationsplattformen und Dashboards haben die Teams jederzeit Zugriff auf alle wichtigen Informationen. Das erleichtert die Abstimmung in Meetings, hilft neuen Teammitgliedern beim schnellen Einarbeiten und macht Probleme frühzeitig sichtbar.

Der gezielte Einsatz dieser Tools kann dazu beitragen, Aufgaben klar zu strukturieren, Abhängigkeiten sichtbar zu machen und die standortübergreifende Zusammenarbeit zu unterstützen. Bei angemessener Nutzung fördern sie Transparenz, erleichtern die Kommunikation sowie den Wissensaustausch und ermöglichen eine genaue Fortschrittskontrolle.

Ein zusätzlicher Vorteil digitaler Tools liegt in der Möglichkeit zur gezielten Suche, Filterung und Kategorisierung von Aufgaben und Informationen. Der gezielte Einsatz solcher Tools stellt daher eine bewährte Best Practice dar. Die folgende Übersicht in Tabelle 20 fasst die wichtigsten funktionalen Anforderungen zusammen, die ein entsprechendes Toolset für agile Teams erfüllen soll:

Funktion	Unterstützte Aspekte
Aufgabenverwaltung	Task-Zuweisung, Kanban-/Sprint-Boards, Statusverfolgung
Transparenz & Fortschrittsmessung	Dashboards, Zeiterfassung, Sprint-Tracking
Kommunikation	Verlinkung mit Meetings und Dokumentation,

	Kommentarfunktionen
Wissensverteilung & Dokumentation	Ablage von technischen Dokumentationen, How-Tos, Sprint-Reviews, Architekturübersichten
Support für verteilte Teams	Standortübergreifender Zugriff, gemeinsame Plattform für alle Rollen
Kategorisierung & Nachvollziehbarkeit	Verlinkung von Aufgaben, Entscheidungen und Dokumenten
Such- und Filterfunktionen	Möglichkeit der Suche und Filterung nach Aufgaben, Status, Kategorien oder Textinhalten

Tabelle 20: Übersicht der Funktionalitäten für den toolunterstützten Workflow

Die tägliche Umsetzung kann wie folgt zusammengefasst werden:

- **Daily Start:**
  Teammitglieder aktualisieren den Aufgabenstatus im Board (z. B. auf „In Bearbeitung“).
- **Daily Meeting:**
  Das Team geht gemeinsam das Sprint- oder Kanban-Board durch, diskutiert Blocker und koordiniert nächste Schritte
- **Während des Tages:**
  Kommentare, Umbuchungen und ergänzende Dokumentation erfolgen direkt im Tool (z.B. Jira, GitLab oder Confluence).
- **Abschluss:**
  Aufgaben werden abgeschlossen, Fortschritt dokumentiert, Review-Notizen ergänzt oder Zeitaufwände eingetragen.

6.1.1.2 Regelmäßige synchrone Abstimmung im Team

Regelmäßige Meetings sind essenziell für die Teamkoordination, um den Arbeitsstand abzugleichen, Abhängigkeiten zu klären und Probleme frühzeitig zu identifizieren. Sie fördern Transparenz, effektive Kommunikation und eine strukturierte Zusammenarbeit. In den Fallbeispielen werden verschiedene Meeting-Formate genutzt, die sich in Frequenz und Fokus unterscheiden.

In allen drei Fallbeispielen gibt es tägliche Abstimmungsrunden in Form von Dailies, in denen Teammitglieder ihre aktuellen Aufgaben, Fortschritte und Blockaden berichten. Diese Meetings dauern in der Regel 15 Minuten, können aber in Ausnahmefällen länger sein. Im Fallbeispiel 2 und 3 werden Jira- oder GitLab-Boards genutzt, um den Status von Tickets („Backlog“, „In Bearbeitung“, „In Review“, „Abgeschlossen“) direkt während der Besprechung zu aktualisieren. Im Fallbeispiel 3 wird zusätzlich die aktive Sprint-Ansicht in Jira genutzt, um den aktuellen Arbeitsstand während des Stand-ups transparent zu machen. Falls Diskussionen den Rahmen sprengen, werden sie nach dem Meeting separat weitergeführt.

> *„Genau also bei Jira über das Sprint-Board. Dort kann man sehen, wer welches Ticket gerade bearbeitet. Also im Normallfall funktioniert es eigentlich ganz okay. Wir haben auch ein Dayli und was man Dayli macht, ist eigentlich mehr ein Reporting. Das heißt, wir gehen genau eben dieses Sprint-Board durch und jeder erzählt, wann er /sie gerade arbeitet und ob es da irgendwelche Probleme gibt. Und irgendwelche dringenden Themen werden halt auch im Dayli besprochen. Aber ansonsten ist es eher mehr so, dass es bleibt bei 15 Minuten also bisschen ge-time-boxed. Im Normalfall bleibt es bei 15 Minuten.“*- T6, Technischer Architekt

Die Abstimmungen sind eng mit den Bereichen Transparenz und Fortschrittsverfolgung verknüpft. Besonders in den Fallbeispielen 2 und 3 werden während der Dailies Statusupdates direkt in Jira oder GitLab gesetzt, um den Arbeitsstand nachvollziehbar zu halten. Im Fallbeispiel 1 und 3 werden wichtige Entscheidungen in Confluence oder internen Wikis dokumentiert, damit auch nicht anwesende Teammitglieder darauf zugreifen können.

Durch die regelmäßige Abstimmung wird sichergestellt, dass alle Teammitglieder stets über den aktuellen Stand informiert sind, Abhängigkeiten frühzeitig erkannt werden und relevante Informationen dokumentiert bleiben. Der Einsatz digitaler Tools wie Jira, GitLab und Confluence unterstützen diesen Prozess und verknüpfen Abstimmungen mit der Aufgabenverfolgung.

Die spezifischen Zielsetzungen und Abgrenzungen von Dailies, Weeklies und rollenspezifischen Teamleiter-Meetings sind in Tabelle 23 zusammengefasst.

6.1.1.3 Wöchentliche Abstimmung zwischen Teams

Neben den Dailies, die sich auf die tägliche Abstimmung innerhalb eines Teams konzentrieren und vor allem den aktuellen Arbeitsfortschritt betreffen, gibt es regelmäßige wöchentliche Meetings (Engl. „Weeklies“), die eine teamübergreifende Koordination ermöglichen. Während Dailies primär operative Abstimmungen sind, dienen Weeklies der längerfristigen Planung und der Synchronisation zwischen autonomen Teams mit gemeinsamen Schnittstellen.
Im Fallbeispiel 1 sind wöchentliche Meetings etabliert, um Fortschritte mit anderen Teams abzugleichen und Abhängigkeiten frühzeitig zu erkennen. Diese Meetings sind vor allem für Teamleiter oder Vertreter mehrerer Teams gedacht, um wichtige Themen gemeinsam zu besprechen und Informationen zwischen den beteiligten Teams und Themen sicherzustellen.

Im Fallbeispiel 2 sind Weeklies fester Bestandteil der Projektorganisation. Neben den Entwicklern nehmen auch Requirements Engineers, Tester oder technische Architekten teil. Diese Meetings dienen nicht nur der Synchronisation zwischen Teams, sondern auch der übergreifenden Abstimmung über priorisierte Aufgaben und strategische Entscheidungen. Darüber hinaus gibt es spezialisierte Meetings, wie etwa Architektur- oder Test-Meetings, die im regelmäßigen Turnus stattfinden.

> *„Ja, natürlich gibt es Meetings, die nach oben Informationen weitergeben. Die Teamleads reden natürlich miteinander, […]“* – T6, Technischer Architekt

Im Fallbeispiel 3 gibt es Weeklies für die Abstimmung mit Projektleitern sowie für das Release- und Test-Management. Diese Meetings haben neben dem Statusabgleich auch die Funktion, Probleme frühzeitig zu identifizieren und gegebenenfalls Eskalationen anzustoßen, wenn Abhängigkeiten zwischen Teams nicht geklärt werden können. Es gibt zudem spezialisierte Meetings mit Teamleitern oder Key-Personen, die strategische Themen diskutieren.

> *„Also, wir haben wöchentliche Tickets durchsprachen mit PM, Release-Managements und Test-Management und da besprechen wir einfach neue Tickets und bewerten diese.“* – T11, Teamleiter

Weeklies sind eine bewährte Praxis, um Transparenz über Teamgrenzen hinweg sicherzustellen und den Informationsfluss zwischen Teams und weiteren relevanten Akteuren zu gewährleisten. Während Dailies sich auf den laufenden Entwicklungsprozess fokussieren, ermöglichen Weeklies eine übergreifende Planung und eine bessere Abstimmung zwischen verschiedenen Fachbereichen. Entscheidend ist, dass diese Meetings strukturiert durchgeführt und die Ergebnisse dokumentiert werden, um die Nachvollziehbarkeit der Abstimmungen zu gewährleisten. Im Unterschied zu den rollenspezifischen Teamleiter-Meetings (siehe Punkt 6.1.7.3), die der fachlichen Koordination innerhalb definierter Rollen dienen, liegt der Fokus der Weeklies auf der operativen Synchronisation zwischen mehreren Teams.

Die spezifischen Zielsetzungen und Abgrenzungen von Dailies, Weeklies und rollenspezifischen Teamleiter-Meetings sind in Tabelle 23 zusammengefasst.

6.1.1.4 Standardisierte Protokollierung und Dokumentation

Eine strukturierte Dokumentation von Meeting-Ergebnissen, Entscheidungen und Verantwortlichkeiten ist essenziell, um Transparenz und Nachvollziehbarkeit in den Teams sicherzustellen.

In den untersuchten Fallbeispielen werden unterschiedliche Methoden genutzt, um sicherzustellen, dass alle relevanten Informationen festgehalten und zugänglich sind.

Im Fallbeispiel 1 erfolgt die Dokumentation größtenteils in internen Wikis, in denen technische Entscheidungen, Sprint-Ergebnisse und Architektur-Überlegungen erfasst werden. In den Fallbeispielen 2 und 3 setzen Teams primär auf das Programm Confluence, um Meeting-Protokolle, Best Practices und Guidelines zu hinterlegen. Besonders im Fallbeispiel 3 wird dies systematisch genutzt, sodass alle Teammitglieder auch nachträglich auf Informationen zugreifen können.

Ein wichtiger Bestandteil dieser Praxis ist die Definition of Ready und Definition of Done, die insbesondere in den Fallbeispielen 2 und 3 festgelegt wurden, um klare Kriterien für die Arbeitsorganisation zu definieren. Im Fallbeispiel 1 gibt es vergleichbare Strukturen zur Qualitätssicherung, etwa in Form einer Review-Checkliste. Diese Definitionen helfen dabei, Missverständnisse über den Fertigstellungsgrad von Aufgaben zu vermeiden und sicherzustellen, dass alle Beteiligten ein gemeinsames Verständnis von Anforderungen und Qualitätssicherung haben.

> *„[...] dass man einfach mal drüber schaut über anderen Code, nach einer bestimmten Checkliste und dann schaut ob das passt und dann halt Review macht.“* – T3, Teamleiter

> *„Wir haben jetzt intern auch eine Definition of Done gemacht, dass stimmt schon, wo drinnensteht eine Userstory muss auch die Testautomatisierung, soweit - soweit als möglich abgedeckt sein. Sie muss getestet sein auf welcher Instanz das sein muss oder so - das haben wir Team intern gemacht“*- T10, Tester

> *„... es soll natürlich und es wird auch so gut wie alles in Confluence dokumentiert. Meeting -Protokolle, technische Themen, How-To's und On-Boarding, Verfügbarkeiten - wird alles in Confluence gemacht.“* – T11, Teamleiter

Durch die einheitliche Protokollierung und Dokumentation bleibt der Informationsfluss konsistent, Verantwortlichkeiten sind nachvollziehbar und auch neue Teammitglieder können sich schnell einarbeiten. Dies verbessert nicht nur die Transparenz im Team, sondern erleichtert auch die langfristige Planung und Entscheidungsfindung. Darüber hinaus leisten definierte Abschlusskriterien wie eine Definition of Done oder strukturierte Review-Checklisten einen wichtigen Beitrag zur Qualitätssicherung, da sie gemeinsame Fertigstellungskriterien schaffen und sicherstellen, dass Aufgaben nicht nur abgeschlossen, sondern auch überprüft und freigegeben wurden.

6.1.2 Kommunikation

Die Codes „Telefon“, „direkte Kommunikation“, „Ticket-Tool“, „Chat“, „Meetings“, „E-Mail“, und „Wiki/Confluence/...“ werden zu folgenden Best Practices ausformuliert:

6.1.2.1 Effiziente Abstimmung durch direkte Kommunikation

Direkte Kommunikation ist ein essenzieller Bestandteil der Zusammenarbeit in den Teams. Sie ermöglicht eine schnelle und effiziente Abstimmung innerhalb und außerhalb des Teams, besonders bei dringenden oder komplexen Themen. Je nach Situation werden verschiedene Kommunikationskanäle genutzt, um Abstimmungen so direkt und effizient wie möglich zu gestalten.

Innerhalb der Teams ist Face-to-Face-Kommunikation oft die bevorzugte Methode, vor allem wenn die Teammitglieder im selben Raum arbeiten. Im Fallbeispiel 1 wird betont, dass direkte Abstimmungen bevorzugt werden, da dies oft schneller ist als formalisierte Meetings. Auch im Fallbeispiel 2 und 3 wird verstärkt auf digitale Kommunikationsmittel wie Chat-Tools oder Telefonate zurückgegriffen.

Telefonate sind in allen Fallbeispielen ein wichtiges Mittel, insbesondere wenn schnelle Klärungen erforderlich sind oder der Austausch über Chats nicht ausreicht. Im Fallbeispiel 1 ist telefonische Abstimmung seltener, da viele Teammitglieder physisch nah beieinandersitzen. Im Fallbeispiel 2 und 3 hingegen wird Telefon häufig genutzt, insbesondere für Abstimmungen mit anderen Teams oder externen Fachabteilungen.

Neben der Face-to-Face- und Telefonkommunikation sind Chat-Tools ein zentraler Kommunikationskanal, da sie sowohl spontane Abstimmungen ermöglichen als auch für die spätere Nachverfolgung von Informationen genutzt werden können. Im Fallbeispiel 1 werden die Programme Rocket.Chat, Cisco Jabber und Lotus Notes eingesetzt, während im Fallbeispiel 2 und 3 primär Microsoft Teams, Rocket.Chat und Jitsi verwendet werden. Diese Tools bieten den Vorteil, dass Nachrichten und Diskussionen nachvollziehbar bleiben und Teammitglieder auch später darauf zugreifen können. In bestimmten Fällen – etwa bei der Klärung technischer Details – wird auch Screensharing über Tools wie Jitsi oder MS Teams genutzt, um Inhalte gemeinsam durchzugehen und Missverständnisse zu vermeiden.
E-Mail wird ebenfalls genutzt, insbesondere für formellere Kommunikation oder wenn es wichtig ist, eine schriftliche Dokumentation von Absprachen zu haben. Im Fallbeispiel 3 wird erwähnt, dass Chat für die meisten internen Abstimmungen genutzt wird, während E-Mail vor allem bei wichtigen Themen oder Eskalationen verwendet wird. Ein weiterer wichtiger Aspekt der direkten Kommunikation ist der schnelle Austausch bei Problemen oder Unsicherheiten. Im Fallbeispiel 3

wird beschrieben, dass Entwickler, wenn ein Ticket zu aufwendig wird oder zu viele Kommentare erfordert, einfach den direkten Austausch suchen – entweder über Chat, einen Anruf oder ein kurzfristiges Meeting.

Neben der täglichen Abstimmung ist direkte Kommunikation auch ein essenzieller Bestandteil der Wissensverteilung und des Supports innerhalb der Teams. Im Fallbeispiel 1 wird betont, dass Teammitglieder ihr Wissen aktiv weitergeben und neue Kollegen durch direkte Gespräche oder kurze Abstimmungen eingearbeitet werden. In den Fallbeispielen 2 und 3 findet ein intensiver Austausch über technische Fragen ebenfalls häufig über direkte Kommunikation statt, da dies schneller und effektiver ist als das Nachlesen von Dokumentationen.

Direkte Kommunikation wird damit nicht nur für den Alltag in agilen Teams, sondern auch als wichtiges Mittel zur schnellen Problemlösung, Wissenstransfer und Abstimmung über Teamgrenzen hinweg genutzt. Die folgende Tabelle 21 zeigt eine Übersicht der eingesetzten Kommunikationswerkzeuge der autonomen Teams mit Vor- und Nachteilen und Beispiele für ihren Einsatz:

Kommunikationsmittel	Zweck / Einsatzbereich	Vorteil	Nachteil	Beispielhafte Verwendung im Team
Face-to-Face	Spontane Abstimmungen im Büro, schnelle Klärung	Persönlich, direkt, schnell	Nicht dokumentiert, nur bei physischer Nähe möglich	Rückfrage zu einem Ticket, informelle Einarbeitung
Telefon	Klärung dringender oder komplexer Fragen, auch remote	Schnell, direkt, auch über Distanz	Keine schriftliche Nachverfolgbarkeit	Abstimmung mit Kollegen im Homeoffice, externe Teams bzw. Projektteilnehmer
Chat	Alltagstauglich für Rückfragen, Koordination, Gruppenkommunikation	Schriftlich, schnell, Teilen von Links/Anhängen möglich	Gefahr von Kontextverlust oder unstrukturierter Kommunikation	Rückfragen im Team, Klärung offener Punkte, Gruppenabsprachen
E-Mail	Formelle Kommunikation, dokumentierte Weitergabe von Infos	Nachvollziehbar, gut dokumentiert, Verteilerfähig, Anhänge/Links möglich	Kann in E-Mail-Flut untergehen, Gefahr, dass sich niemand direkt angesprochen fühlt	Weiterleitung von Entscheidungen, Ankündigungen, Abstimmung mit Leitung oder Fachabteilungen
Videocall mit Screensharing	Gemeinsames Durchgehen technischer oder visueller Themen	Visualisierung, direkte Rücksprache, Screensharing möglich	Höherer Koordinationsaufwand, nicht immer dokumentiert	Gemeinsame Analyse eines Problems, Pair Debugging

Tabelle 21: Kommunikationsmittel autonomer Teams

6.1.2.2 Toolunterstützter Workflow

Die Best Practice „Toolunterstützter Workflow" wird bereits im Punkt 6.1.1.1 beschrieben und deckt auch einen Teil der Kategorie „Kommunikation" ab.

6.1.2.3 Regelmäßig synchrone Abstimmung im Team

Die Best Practice „Regelmäßige synchrone Abstimmung im Team" wird bereits im Punkt 6.1.1.2 beschrieben und deckt auch einen Teil der Kategorie „Kommunikation" ab.

6.1.3 Reflexion & Lernprozess

Der Code „Retrospektive & Review" wird zu folgenden Best Practices ausformuliert:

6.1.3.1 Regelmäßige Reflexion und Feedbackschleifen

Retrospektiven sind ein zentrales Instrument in agilen Teams, um kontinuierliche Verbesserungen voranzutreiben. Sie ermöglichen es, Erfahrungen aus abgeschlossenen Sprints zu reflektieren, Probleme zu identifizieren und gezielte Maßnahmen zur Optimierung der Arbeitsweise zu entwickeln. In den untersuchten Fallbeispielen kommen Retrospektiven regelmäßig zum Einsatz, jedoch mit unterschiedlichen Ansätzen und Herausforderungen.

Im Fallbeispiel 1 werden Retrospektiven zwar durchgeführt, es fehlt jedoch manchmal an der konsequenten Umsetzung der erarbeiteten Maßnahmen. Teams besprechen Probleme und Verbesserungspotenziale, aber die Umsetzung der geplanten Änderungen ist nicht immer durchgängig gewährleistet. In den Fallbeispielen 2 und 3 hingegen sind Retrospektiven fester Bestandteil des Entwicklungsprozesses und werden aktiv genutzt, um nach jedem Sprint konkrete Verbesserungsmaßnahmen abzuleiten.

Besonders im Fallbeispiel 3 haben Teams eine strukturierte Herangehensweise entwickelt, indem „Mantras" in Confluence dokumentiert werden. Diese langfristigen Leitlinien basieren auf den Erkenntnissen aus Retrospektiven und helfen, wiederkehrende Herausforderungen gezielt anzugehen. Zusätzlich werden konkrete Actions für den nächsten Sprint definiert, um sicherzustellen, dass Verbesserungsvorschläge nicht nur diskutiert, sondern auch umgesetzt werden.

Neben klassischen Sprint-Retrospektiven gibt es im Fallbeispiel 2 auch technische Reviews, bei denen Entwickler alle zwei Wochen gemeinsam Code-Qualitätsthemen wie SonarQube-Defects oder Architekturentscheidungen analysieren. Dadurch entstehen regelmäßige Feedbackeinheiten, die nicht nur die Entwicklungsprozesse optimieren, sondern auch als Wissenstransfer innerhalb der Teams dienen.

Durch die regelmäßige Reflexion und strukturierte Dokumentation stellen Teams sicher, dass sie aus vergangenen Fehlern lernen, ihre Prozesse kontinuierlich verbessern und nachhaltige Veränderungen etablieren. Retrospektiven tragen somit nicht nur zur Prozessoptimierung, sondern auch zur langfristigen Teamkultur und Produktivitätssteigerung bei.

6.1.3.2 Pair-Programming, 4-Augen-Prinzip und Reviews

Code-Reviews sind ein essenzieller Bestandteil der Qualitätssicherung und Wissensverteilung in Softwareentwicklungsprojekten. Besonders in Softwaregroßprojekten mit mehreren Teams sorgen strukturierte Review-Prozesse dafür, dass Fehler frühzeitig erkannt, Best Practices etabliert und Wissen über Teamgrenzen hinweg geteilt wird. Eine zentrale Praxis in den Fallbeispielen ist das 4-Augen-Prinzip, das sowohl bei teaminternen als auch teamübergreifenden Reviews angewendet wird.

Im Fallbeispiel 1 ist das 4-Augen-Prinzip nicht immer verpflichtend, wird aber in vielen Teams genutzt, um sicherzustellen, dass komplexere Änderungen von einer zweiten Person überprüft werden. Gerade bei größeren Tasks wird oft vor dem Commit eine Review durch eine andere Person durchgeführt, um mögliche Probleme frühzeitig zu erkennen. Im Fallbeispiel 2 sind Reviews über Merge-Requests fester Bestandteil des Entwicklungsprozesses. Im Fallbeispiel 3 ist das Prinzip weiter formalisiert – es gibt eine klare Vorgabe, dass kein Entwickler seinen eigenen Code freigeben darf, sondern dieser immer von einem anderen Teammitglied überprüft werden muss. Jede Änderung muss durch eine zweite Person geprüft werden, bevor sie in den Master-Branch übernommen wird.

Neben teaminternen Reviews sind in den Fallbeispielen 2 und 3 auch teamübergreifende Reviews notwendig, insbesondere wenn mehrere Teams an einer gemeinsamen Codebasis arbeiten. Im Fallbeispiel 2 gibt es beispielsweise Module, die unter der Verantwortung eines bestimmten Teams stehen. Änderungen daran müssen erst von diesem Team geprüft und freigegeben werden. Ähnlich ist es im Fallbeispiel 3, wo teamübergreifende Code-Reviews sicherstellen, dass gemeinsame Standards eingehalten und unerwartete Seiteneffekte vermieden werden.

Zusätzlich zur klassischen Code-Review-Praxis kommen in den Fallbeispielen Pair-Programming, Screen-Sharing und Review-Meetings zum Einsatz. Besonders im Fallbeispiel 1 werden Reviews oft informell während der Entwicklung durchgeführt, indem sich Entwickler zu komplexen Themen direkt abstimmen oder gemeinsam an einem Code-Abschnitt arbeiten. Im Fallbeispiel 2 wird beschrieben, dass Review-Prozesse nicht nur zur Fehlererkennung, sondern auch als Möglichkeit genutzt werden, alternative Lösungswege zu diskutieren und voneinander zu lernen.

Neben der Qualitätssicherung trägt das 4-Augen-Prinzip auch dazu bei, komplexe Aufgabenbereiche besser zu lösen. In den Fallbeispielen 2 und 3 werden besonders schwierige oder technisch anspruchsvolle Änderungen oft in Pair-Programming-Sessions oder durch intensive Reviews bearbeitet, um verschiedene Perspektiven einzubeziehen und Lösungen effizienter zu entwickeln. Im Fallbeispiel 3 ist es üblich, dass sich zwei Entwickler gemeinsam eine komplexe Implementierung ansehen, bevor der Code weitergeführt wird. Dadurch werden Fehlentscheidungen vermieden, Architekturanforderungen gezielt berücksichtigt und alternative Lösungswege systematisch diskutiert.

> *„Also für die ganz komplexen Sachen versuchen wir uns vorher in drei, vier, fünf Leuten vielleicht nochmal zusammen zu reden, die sich mit dem Thema halbwegs auskennen, um zu evaluieren, wie wir das umsetzen können. Ansonsten, wenn's nur etwas komplexer ist, dann geht's Richtung Pair-Programming."* – T3, Teamleiter

> *„Also Pair-Programming ist eine Option - wird primär eingesetzt, wenn jemand sich natürlich überhaupt nicht auskennt in einem Thema aber da halt hineinkommen möchte. Also vor allem Neuzugängen im Team. Dann wird es eher eingesetzt."* – T6, Technischer Architekt

Die übergreifende Best Practice ist der gezielte Einsatz von 4-Augen-Konstellationen – sei es als Pair-Programming, formelle Reviews oder spontane gemeinsame Problemanalyse. Diese Arbeitsweise vereint mehrere Ziele gleichzeitig: Sie sichert Qualität, fördert Wissensaustausch, erleichtert Einarbeitung, stärkt gemeinsame Verantwortung und ermöglicht kontinuierliches Lernen im Team.

6.1.4 Wissensverteilung & Support

Die Codes „Code Review", „Dokumentation", „Pair-Programming", „Taskrotation", „Direkte Nachfrage (Sms, Telefon)" und „Mehrere Personen pro Thema" werden zu folgenden Best Practices ausformuliert:

6.1.4.1 Pair-Programming, 4-Augen-Prinzip und Reviews

Die Best Practice „Pair-Programming, 4-Augen-Prinzip und Reviews" wird bereits im Punkt 6.1.3.2 beschrieben und deckt auch einen Teil der Kategorie „Wissensverteilung & Support" ab.

6.1.4.2 Toolunterstützter Workflow

Die Best Practice „Toolunterstützter Workflow" wird bereits im Punkt 6.1.1.1 beschrieben und deckt auch einen Teil der Kategorie „Wissensverteilung & Support" ab.

6.1.4.3 Effiziente Abstimmung durch direkte Kommunikation

Die Best Practice „Effiziente Abstimmung durch direkte Kommunikation" wird bereits im Punkt 6.1.2.1 beschrieben und deckt auch einen Teil der Kategorie „Wissensverteilung & Support" ab.

6.1.4.4 Mehrere Themenverantwortliche zur Wissensverteilung

Um sicherzustellen, dass Wissen nicht nur bei einzelnen Spezialisten konzentriert ist, setzen die Teams in den Fallbeispielen darauf, dass mehrere Personen für ein Themengebiet verantwortlich sind. Diese Strategie hilft, Wissensinseln zu vermeiden, die Teamflexibilität zu erhöhen und sicherzustellen, dass Aufgaben auch bei Abwesenheiten oder Personalwechseln problemlos weitergeführt werden können.

In allen drei Fallbeispielen gibt es bewusste Maßnahmen zur Verteilung von Verantwortlichkeiten. Im Fallbeispiel 1 ist dies zwar das erklärte Ziel, jedoch wird es in den Teams unterschiedlich konsequent umgesetzt. Im Fallbeispiel 2 werden zentrale Themen gezielt auf mehrere Personen aufgeteilt, sodass immer mindestens eine zweite Person mit dem Gebiet vertraut ist. Im Fallbeispiel 3 ist die Wissensteilung stark in die tägliche Zusammenarbeit integriert – Teammitglieder stimmen sich regelmäßig ab und tauschen Informationen aus, um sicherzustellen, dass nicht nur Einzelne über entscheidende Details Bescheid wissen.

> *„Jetzt ist es so wir versuchen natürlich, soweit es geht, zumindest zwei Personen pro Thema zu haben, die sich auskennen."* – T6, Technischer Architekt

Ein bewährtes Mittel zur Wissensverteilung ist die gezielte Rotation von Aufgaben (Task-Rotation), die in den Fallbeispielen in unterschiedlicher Intensität genutzt wird. Im Fallbeispiel 1 werden Aufgaben so verteilt, dass Teammitglieder verschiedene Bereiche kennenlernen und nicht nur

an einem einzigen Themengebiet arbeiten. Im Fallbeispiel 2 übernehmen Entwickler gezielt auch Aufgaben außerhalb ihrer bisherigen Spezialisierung, um sich in neue Bereiche einzuarbeiten. Im Fallbeispiel 3 wird eine Mischung aus Spezialisierung und Rotation praktiziert: Während einige Teammitglieder feste Schwerpunkte haben, wird darauf geachtet, dass sie auch an anderen Themen mitarbeiten.

> *„Ich - ich denke, man versucht es halt, dass man halt die Themen auch durchmischt. Das halt jemand mal ein Ticket her nimmt […] er sich natürlich beschäftigt hat mit diesem Teil oder Komponente, dass man es so quasi rotiert, […]. Aber ich denke, das ist einer der wenigen Möglichkeiten, dass man versucht, keine Wissensinseln aufzubauen, indem man halt einfach jedem jedes Ticket gibt, auch wenn es mal länger dauert ein bisschen."* – T11, Teamleiter

Ein typisches Beispiel für diese Praxis ist die Übernahme von Unit-Tests oder funktionalen Erweiterungen an bestehenden Features. Im Fallbeispiel 3 wird explizit darauf geachtet, dass keine Person allein für ein Thema zuständig ist.

In allen drei Fallbeispielen zeigt sich als bewährte Best Practice die gezielte Kombination aus Mehrfachbesetzung von Themengebieten und rotierender Verteilung der Aufgaben. Durch die bewusste Verteilung der Arbeit wird sichergestellt, dass Teams nicht von einzelnen Schlüsselpersonen abhängig sind und neue Teammitglieder schrittweise in unterschiedliche Themen eingearbeitet werden können. Diese Vorgehensweise ermöglicht eine nachhaltige Wissensverteilung, reduziert das Risiko von Wissensinseln und erhöht zugleich die Flexibilität und Weiterentwicklungsmöglichkeiten innerhalb des Teams.

Diese Praktik unterscheidet sich vom gezielten Themenwechsel (siehe Punkt 6.1.5.2), da der Fokus hier auf der parallelen Verantwortlichkeit mehrerer Teammitglieder pro Thema liegt – unabhängig von individuellen Weiterentwicklungswünschen.

6.1.5 Fortbildung

Die folgenden Best Practices leiten sich aus den Codes „Zertifikate & Weiterbildung (aktiv)", „Zeit für Verbesserungen", „Themenwechsel", „Rollenwechsel" und „Zeit für Technologiewechsel & Updates" ab. Tabelle 22 gibt einen kompakten Überblick über die Unterschiede zwischen diesen Best Practices und zeigt deren jeweilige Schwerpunkte, Umsetzung und Ziele auf.

6.1.5.1 Weiterbildung durch Eigeninitiative und Teamförderung

In allen drei Fallbeispielen besteht grundsätzlich die Möglichkeit zur individuellen Weiterbildung, etwa durch Schulungen, Zertifikate oder die Beschäftigung mit neuen Technologien. In keinem der drei Fallbeispiele gibt es eine klar geplante oder abgestimmte Vorgehensweise für Fortbildungsmaßnahmen. Stattdessen hängt die tatsächliche Umsetzung stark von der Eigeninitiative der Mitarbeitenden ab. Wer sich aktiv um eine Weiterbildung bemüht, erhält meist Unterstützung durch die Teamleitung, sei es durch zeitliche Freiräume, formale Genehmigung oder der Kostenübernahme.

Im Fallbeispiel 1 gibt es keine festen Programme oder zentralen Angebote. Mitarbeitende müssen von sich aus den Wunsch zur Weiterbildung einbringen. Zertifizierungen wie Scrum Master oder sicherheitsbezogene Schulungen sind grundsätzlich möglich, müssen aber selbst vorgeschlagen und organisiert werden.

> *„Aber man muss halt pro-aktiv agieren, also es ist nicht so dass ein Organisator herkommt und sagt: Du jetzt schau dir mal diese oder jene Technologie an und schau ob es da Verbesserungsmöglichkeiten gibt, sondern - z.B. in Java, wenn du ein Zertifikat machen willst dann musst halt du nachfragen, aktiv.“* – T2, Softwareentwickler

Auch im Fallbeispiel 2 ist ebenfalls eine Weiterbildung möglich, aber mit organisatorischen Hürden verbunden. Für eigenständige Lernphasen muss zunächst ein formales Analyse-Ticket erstellt werden, das dann in die Projektplanung aufgenommen werden kann. In der Praxis ist es daher häufig nur dann realistisch, wenn es im Rahmen bestehender Aufgaben geschieht oder zeitliche Freiräume zufällig entstehen.

Fallbeispiel 3 zeigt die offenste Herangehensweise: Hier wird Weiterbildung aktiv unterstützt, sofern sie für die Projektarbeit relevant ist. Es gibt regelmäßige Reflexionsgespräche zu Karrierewünschen, Zeitfenster zur individuellen Erkundung technischer Themen sowie konkrete Schulungen etwa im Bereich IT-Security. Auch formale Zertifizierungen werden ermöglicht, wenn sie zur Weiterentwicklung beitragen.

> *„Das ist in der Regel schon so, dass jeder Entwickler auch ein Zeitfenster hat, wo er sich weiterbilden kann.“* – T12, Technischer Architekt

Eine zentrale Herausforderung in allen drei Fällen ist es, Lernen mit den laufenden Projektverpflichtungen zu vereinbaren. Dennoch zeigt sich als bewährte Praxis in allen drei Fallbeispielen:

Weiterbildung der Mitglieder wird gefördert, wenn sie aus Eigeninitiative angestoßen wird und in sich mit dem Projektablauf vereinbaren lässt.

Eine vergleichende Übersicht der Praktiken zur Fortbildung ist in Tabelle 22 dargestellt.

6.1.5.2 Themenwechsel zur individuellen Weiterentwicklung

Themenwechsel werden in den untersuchten Teams gefördert, um Wissen breiter zu streuen und Mitarbeitenden die Möglichkeit zu geben, sich fachlich weiterzuentwickeln. Der Wechsel wird meist von den Teammitgliedern selbst angestoßen. Gleichzeitig profitiert auch das Team davon, da eine breitere Verteilung von Kompetenzen dazu beiträgt, Wissensinseln zu vermeiden und Ausfälle besser abzufangen. Im Unterschied zu Punkt 6.1.4.4 steht hier die individuelle Motivation zum Themenwechsel im Vordergrund, die mit Unterstützung des Teams realisiert wird.

Im Fallbeispiel 1 gibt es keine festgelegte Rotation, aber wer Interesse an einem neuen Thema hat, kann sich aktiv dafür einsetzen. Häufig beginnt der Wechsel mit kleineren Aufgaben, bevor ein vollständiger Übergang erfolgt. Unterstützung aus dem Team ist in der Regel gegeben, sofern es mit den Projektzielen vereinbar ist.

> *„Wenn man wirklich was anderes machen will dann muss man sagen man will in ein anderes Teilprojekt, man muss selber aktiv sein.“* - T1, Requirements Engineer, Tester und Softwareentwickler

Im Fallbeispiel 2 ist der Themenwechsel strukturierter möglich. Teammitglieder können sich gezielt in neue Bereiche einarbeiten, insbesondere wenn dadurch Wissensverteilung gefördert oder Engpässe vermieden werden. Die Abstimmung erfolgt mit dem Teamleiter und den zuständigen Kolleginnen, um einen reibungslosen Übergang zu gewährleisten.
Im Fallbeispiel 3 sind Themenwechsel flexibel gestaltet und basieren auf der Initiative der Teammitglieder. Wer sich für ein neues Gebiet interessiert, bringt dies in Meetings oder Gesprächen ein und übernimmt nach und nach entsprechende Aufgaben. Der Wechsel erfolgt schrittweise und in Abstimmung mit dem Team, sodass Wissen gezielt aufgebaut und integriert werden kann.

> *„Ich hätte das eher damit beantwortet, dass wenn jemand woanders hinwill, dann sagt er es und dann schaut man, dass man eine Lösung findet.“* - T14, Projektleiter

Themenwechsel sind in allen untersuchten Teams möglich – entweder wenn sie von Mitarbeitenden eigeninitiativ angestoßen werden oder wenn sie aus projektfachlicher Sicht sinnvoll erscheinen. Als etablierte Praxis hat sich gezeigt, dass Wechselwünsche offen kommuniziert, schrittweise umgesetzt und vom Team aktiv begleitet werden. Auf diese Weise lässt sich individuelles Interesse mit den Anforderungen des Projekts verbinden – zum Nutzen sowohl der persönlichen Entwicklung als auch der nachhaltigen Wissensverteilung im Team. Eine vergleichende Übersicht der zur Fortbildung ist in Tabelle 22 dargestellt.

6.1.5.3 Rollenwechsel zur Entwicklung individueller Fähigkeiten

Rollenwechsel innerhalb eines Projekts bieten Teammitgliedern die Möglichkeit, neue Kompetenzen zu entwickeln und ihre Einsatzfähigkeit im Team zu erweitern. Während Themenwechsel innerhalb der bestehenden Rolle stattfinden, geht ein Rollenwechsel mit einer veränderten Verantwortung einher, beispielsweise der Wechsel von der Entwicklung zum Test oder vom Backend zu DevOps-Themen. Besonders in autonomen Teams bringt dies den Vorteil, dass Teammitglieder flexibel mehrere Rollen übernehmen oder bei Engpässen in anderen Bereichen aushelfen können, wodurch Wissen besser verteilt und Abhängigkeiten reduziert werden. In den untersuchten Fallbeispielen zeigt sich, dass Rollenwechsel grundsätzlich möglich sind, aber meist von den Mitarbeitenden selbst angestoßen werden und von den jeweiligen Anforderungen im Team abhängen.

Im Fallbeispiel 1 gibt es keine festgelegte Regelung für Rollenwechsel, aber Teammitglieder können in neue Rollen hineinwachsen, sofern dies sinnvoll erscheint. Meist erfolgt dies schrittweise, indem zunächst Aufgaben aus der neuen Rolle übernommen werden, bevor ein vollständiger Wechsel stattfindet. Wenn beispielsweise ein Entwickler Interesse am Testen zeigt, kann er erste Testfälle erstellen und sich in Abstimmung mit dem Teamleiter tiefer in das Thema einarbeiten.

> *„Also es gibt für die internen Entwickler schon die Möglichkeit, dass sie sich Abteilungsmäßig versetzen lassen. Das kann dann auch sein, dass ein Kollege von der Softwareentwicklung in den Test Bereich wechselt und ab dann aber wirklich nur noch testet.“* – T5, Teamleiter

Im Fallbeispiel 2 wird Rollenwechsel aktiver unterstützt, insbesondere wenn dies zur besseren Verteilung von Wissen und Verantwortlichkeiten beiträgt. Viele Teammitglieder haben über die Zeit ihre Schwerpunkte erweitert, um neue Fähigkeiten aufzubauen und das Team flexibler aufzustellen. Der Wechsel erfolgt hier oft in Absprache mit dem Team und richtet sich danach, wo Unterstützung gebraucht wird.

Im Fallbeispiel 3 gibt es ebenfalls keine festen Vorgaben, aber eine Offenheit für Rollenwechsel, wenn dies von den Teammitgliedern gewünscht wird. Wer sich für einen Wechsel interessiert, bringt dies meist in Meetings zur Sprache und übernimmt nach und nach Aufgaben aus der neuen Rolle. Der Übergang erfolgt schrittweise, wobei das Team unterstützt und Wissen aktiv weitergegeben wird.

> *„Selbstverständlich. Wenn Teammitglieder gerne wo anders hinwollen, dann können sie es kommunizieren und es wird nach Möglichkeit berücksichtigt.“* – T15, technischer Architekt und Softwareentwickler

Rollenwechsel sind eine bewährte Praxis, um Fachwissen innerhalb des Teams zu erweitern und die individuelle Entwicklung der Mitarbeitenden zu fördern. In der Praxis zeigt sich jedoch, dass Rollenwechsel in den meisten Teams nicht systematisch gesteuert, sondern vor allem durch die Eigeninitiative der Mitarbeitenden angestoßen werden. Während in einzelnen Fällen gezielte Wechsel stattfinden, beruhen die Übergänge überwiegend auf persönlichem Interesse und situativen Bedürfnissen im Team. Entscheidend ist, dass ausreichend Unterstützung geboten wird, damit der Wechsel reibungslos gelingt und langfristig zur Teamstabilität beiträgt.

Kriterium	6.1.5.1 Weiterbildung durch Eigeninitiative und Teamförderung	6.1.5.2 Themenwechsel zur individuellen Weiterentwicklung	6.1.5.3 Rollenwechsel zur Entwicklung individueller Fähigkeiten
Fokus	Gezielte Schulungen, Zertifizierungen und Zeit für technologische Erprobung	Übernahme neuer Themengebiete innerhalb des Projekts zur Kompetenzentwicklung	Wechsel in eine neue fachliche Rolle, z. B. Entwickler zu Tester, Backend zu DevOps
Umsetzungsform	Teilnahme an Kursen, Zertifikaten, interne Schulungen, technologische Tests in Lernphasen	Mitarbeitende übernehmen neue Themen schrittweise und in Abstimmung mit dem Team	Schrittweise Einarbeitung in neue Rollen, oft abhängig von Bedarf oder persönlichem Interesse
Beteiligte	Teammitglieder, Teamleiter, externe Schulungsanbieter	Teammitglieder, Teamleiter, Fachverantwortliche	Teammitglieder, Teamleiter, Projektleitung
Ziel	Fachliche Weiterentwicklung, Sicherstellung technologischer Kompetenz, Innovationsförderung	Wissensverteilung, Flexibilität erhöhen, Vermeidung von Wissensinseln	Erweiterung der Fähigkeiten, Reduktion von Abhängigkeiten, Flexibilität im Team verbessern

Tabelle 22: Unterscheidung der Praktiken für die Fortbildung

6.1.6 Richtlinien, Grenzen, Normen und Prinzipien

Folgende Best Practices ergeben sich aus den Codes „Doku (Wiki/Confluence/...)“, „Organisationsweite Richtlinien“, „Festlegung zentral durch Spezialisten (1...n)“ und „zusätzliche Codequalitäten durch Team bzw. Teamleiter“:

6.1.6.1 Dokumentation und Festlegung von Richtlinien und Prozessen

In Softwaregroßprojekten mit vielen autonomen Teams ist eine klare Dokumentation von Entwicklungsrichtlinien essenziell, um einheitliche Standards sicherzustellen und die Zusammenarbeit zwischen Teams zu erleichtern. Ohne zentrale Vorgaben würden sich unterschiedliche Herangehensweisen entwickeln, was zu Inkonsistenzen in Architektur, Code-Qualität und Entwicklungsprozessen führen könnte. In allen drei Fallbeispielen existieren etablierte Strukturen, die gewährleisten, dass technische und organisatorische Vorgaben definiert, dokumentiert und für alle Beteiligten zugänglich sind. Diese Richtlinien werden nicht von einzelnen Teams autonom festgelegt, sondern durch spezialisierte Rollen oder koordinierende Einheiten betreut. Im Fallbeispiel 1 existiert ein entwicklungsinternes Wiki, in dem Architekturentscheidungen, Coding-Guidelines und Tool-Vorgaben dokumentiert sind. Die Verantwortung für deren Festlegung liegt bei technischen Architekten sowie einem Software-Entscheidungsgremium (SEG), das zentrale Standards vorgibt.

> *„Also grundsätzlich gibt es in dem Projekt einen technischen Architekten, der dafür zuständig wäre, übergreifend für, für alle Entwickler Richtlinien, Entwicklungsvorgehen, Richtlinien, Prinzipien, Architektur, Entscheidungen, Tools usw. vorzugeben und eigentlich auch zu schauen, dass das dann entsprechend umgesetzt wird.“* – T4, Softwareentwickler und Tester

Im Fallbeispiel 2 arbeiten die Teams auf Basis organisationsweiter technischer Vorgaben, die projektübergreifend gelten und zentral dokumentiert sind. Diese beinhalten unter anderem Richtlinien zur grafischen Oberfläche, Konventionen für Schnittstellendefinitionen sowie die Nutzung definierter Werkzeuge. Die Informationen werden beispielsweise im unternehmensweiten Confluence-Wiki dokumentiert und in Meetings sowie Architekturabsprachen weiterentwickelt.

> *„Es gibt aber natürlich auch die grafische Oberfläche. Da gibt es eine Vorgabe, die ist aber global für die Gesamtheit damit das halt von allen Teams richtig umgesetzt wird.“* – T10, Tester

Im Fallbeispiel 3 gelten unternehmensweit verbindliche Sicherheits- und Qualitätsvorgaben, die für alle Teams und Projekte verpflichtend einzuhalten sind. So müssen beispielsweise sämtliche eingesetzten Entwicklungswerkzeuge vor ihrem Einsatz sicherheitstechnisch geprüft und offiziell freigegeben werden.

> *„[…] dieser Prozess ist ziemlich nah dran an dem, was da auf diesen Ticket Checklisten drinnen steht. Also dass man sagt, bevor was umgesetzt wird, muss es sicherheitstechnisch angeschaut worden sein und dann in der Umsetzung darf es nur mit*

> *bestimmten Tools verwendet werden. Alle Entwicklungswerkzeuge müssen Security geprüft sein und freigegeben worden sein.“* – T14, Projektleiter

Die konsequente Kombination aus klar zugewiesenen Verantwortlichkeiten und systematisch gepflegter Dokumentation hat sich in allen drei Fallbeispielen als effektive Best Practice erwiesen. Sie sorgt dafür, dass zentrale Richtlinien nicht nur formal existieren, sondern im Entwicklungsalltag aktiv genutzt und fortlaufend weiterentwickelt werden. Gleichzeitig bleibt Raum für teamspezifische Ergänzungen, wodurch sowohl Einheitlichkeit als auch Flexibilität in großen Projekten gewährleistet werden. Dies erleichtert die Zusammenarbeit zwischen Teams, verbessert die Einarbeitung neuer Mitglieder und trägt maßgeblich zur Erhaltung der Softwarequalität bei.

6.1.6.2 Teamspezifische Erweiterungen von Qualitätsstandards

Neben zentral definierten Richtlinien haben viele Teams in den Fallbeispielen eigene zusätzliche Qualitätsanforderungen entwickelt, die über die allgemeinen Vorgaben hinausgehen. Diese teamspezifischen Standards ergänzen die organisationweiten Coding-Guidelines und helfen dabei, die Code-Qualität langfristig zu verbessern.

Im Fallbeispiel 1 setzen einige Teams gezielt auf Zusatzanforderungen für Testabdeckung und Code-Qualität. Teamleiter können zusätzliche Qualitätsrichtlinien durchsetzen und deren Einhaltung kontrollieren, um sicherzustellen, dass einheitliche Standards eingehalten werden. Dies betrifft insbesondere Anforderungen an die Test Coverage, die als Maß für die Qualität neuer Implementierungen dient.

> *„Und ansonsten haben die Teamleiter sag ich mal noch die Freiheit, dass der zusätzliche Codequalitäten erzwingen können und überprüfen, dass sie auch eingehalten werden, wie zum Beispiel Test Coverage und so Sachen.“* – T3, Teamleiter

Im Fallbeispiel 2 wird beschrieben, dass bestimmte Code-Standards flexibel auf Teamebene festgelegt werden können, solange sie den organisationsweiten Vorgaben nicht widersprechen. Entwickler haben die Möglichkeit, spezifische Guidelines für Code-Reviews oder Modulstrukturen zu definieren, die an ihre Arbeitsweise angepasst sind. Im Fallbeispiel 3 sind Checklisten für Entwicklungsprozesse ein etabliertes Vorgehen, um sicherzustellen, dass alle wichtigen Qualitätskriterien eingehalten werden. Diese Listen enthalten Vorgaben zu Code-Reviews, Testanforderungen und Dokumentationspflichten, die geprüft werden müssen, bevor eine Aufgabe als „Done“ markiert wird. Der Einsatz solcher Checklisten sorgt dafür, dass auch weniger erfahrene Teammitglieder einen klaren Leitfaden für qualitativ hochwertigen Code haben.

Ein konkretes Beispiel für eine dokumentierte und gelebte Richtlinie ist in den Fallbeispielen 2 und 3 die Definition of Done. Sie legt verbindlich fest, wann eine Aufgabe als abgeschlossen gilt, und umfasst Kriterien wie Testabdeckung, Sicherheitsanforderungen oder Reviewpflicht. Neben ihrer Funktion als Qualitätsstandard spielt sie – wie bereits in Kapitel 6.1.1.4 im Kontext transparenter Dokumentation beschrieben – auch eine zentrale Rolle für die Nachvollziehbarkeit im Team.

> *„Wir haben auch noch eine Definition of Ready bzw. eine Definition of done - die haben wir uns auch selber angelegt im Confluence. Und sonstige Richtlinien leben halt in der Knowledge-World, wie es bei […] heißt. Es ist auch nichts weiter als ein eigener Confluence-Bereich."* – T9, Teamleiter und Requirements Engineer

> *„Wir haben jetzt intern auch eine Definition of Done gemacht, dass stimmt schon, wo drinnen steht eine Userstory muss auch die Testautomatisierung, soweit - soweit als möglich abgedeckt sein. Sie muss getestet sein auf welcher Instanz das sein muss oder so - das haben wir Team intern gemacht."* – T10, Tester

> *„[…] gibt es ein paar Checks bevor es gebaut wird. Dann wird es gebaut und dann gibt es einige Checks dafür - wann ist es Done? Typische Definition of Done […]"* – T14, Projektleiter

Zusätzlich setzen viele Teams in den untersuchten Fallbeispielen auf automatisierte Tools wie zum Beispiel SonarQube oder Checkstyle, um Verstöße gegen die definierten Standards frühzeitig zu erkennen. Diese Tools analysieren den Code auf Fehlermuster, Formatierungsabweichungen und strukturelle Probleme und geben Entwicklern direktes Feedback.

> *„Ansonsten haben wir bei uns intern SonarQube, zur Verwendung von dynamischer Code Analyse und auch Checkstyle, wobei wir als Team selbst sehr autonom in unserer Umsetzung sind. Das heißt, bei uns, bei uns setzen sich die Kollegen vom Entwicklungsteam in regelmäßigen Abständen zusammen und gehen die Sonar-Issues durch […]"* – T9, Teamleiter und Requirements Engineer

Die Kombination aus zentralen Vorgaben und teamspezifischen Erweiterungen stellt eine gelebte Best Practice dar, die in allen drei Fallbeispielen fest verankert ist. Autonome Teams nutzen systematisch zusätzliche Qualitätsrichtlinien, strukturierte Checklisten sowie automatisierte Tools wie SonarQube oder Checkstyle, um die Einhaltung definierter Standards sicherzustellen und eigene Anforderungen flexibel zu ergänzen. Dieses Zusammenspiel erlaubt es den Teams, die

Code-Qualität kontinuierlich zu verbessern und an ihre spezifischen Kontexte anzupassen – ohne dabei die übergreifende Konsistenz im Gesamtprojekt zu gefährden.

6.1.7 Meetings

Die folgenden Best Practices leiten sich aus den Codes „Weekly (1-2 Wochen Zyklus)“, „Dailies“ und „Teamleiter-Meetings (Test, Architektur, etc.)“ ab. In der Tabelle 23 gibt einen kompakten Überblick über die Unterschiede zwischen diesen drei Meeting-Typen und zeigt deren jeweilige Schwerpunkte, Teilnehmer und Ziele auf.

6.1.7.1 Regelmäßige synchrone Abstimmung im Team

Die Best Practice „Regelmäßige synchrone Abstimmung im Team” wird bereits im Punkt 6.1.1.2 beschrieben und deckt auch einen Teil der Kategorie „Meetings“ ab.

6.1.7.2 Wöchentliche Abstimmung zwischen Teams

Die Best Practice „Wöchentliche Abstimmung zwischen Teams” wird bereits im Punkt 6.1.1.3 beschrieben und deckt auch einen Teil der Kategorie „Meetings“ ab.

6.1.7.3 Rollenspezifische Teamleiter-Meetings zur fachlichen Koordination

Neben den regelmäßigen Abstimmungen innerhalb der Teams (Dailies) und der teamübergreifenden Koordination in den Weeklies gibt es in den untersuchten Fallbeispielen rollenspezifische Teamleiter-Meetings, die der gezielten Abstimmung zwischen Verantwortlichen für bestimmte Fachbereiche dienen. Diese Meetings sind darauf ausgerichtet, technische, organisatorische oder strategische Themen innerhalb spezifischer Rollen zu besprechen und teamübergreifende Herausforderungen effizient zu koordinieren.

Im Fallbeispiel 1 gibt es seit etwa einem Jahr wöchentliche Meetings der Teamleiter, bei denen zentrale Themen besprochen und der Status ausgetauscht wird. Diese Treffen dienen vor allem dazu, übergreifende Herausforderungen zu identifizieren und Entscheidungen auf einer höheren Ebene zu treffen. In diesen Meetings sind neben den Teamleitern auch Personen mit weiterführenden Entscheidungsbefugnissen beteiligt, sodass Eskalationen direkt adressiert werden können.

Im Fallbeispiel 2 finden spezialisierte Meetings für unterschiedliche Fachbereiche statt. Technische Architekten treffen sich regelmäßig, um Architekturentscheidungen abzustimmen, während es separate Meetings für Testthemen und Requirements Engineering gibt. Diese Meetings ermöglichen eine gezielte Abstimmung innerhalb der jeweiligen Rollen und sorgen für eine klare Struktur bei der Entscheidungsfindung zu spezifischen Fachthemen.

> *„Ja, natürlich gibt es Meetings, die nach oben Informationen weitergeben. Die Teamleads reden natürlich miteinander, es gibt dann auch Test Meetings, wo dann halt immer wieder auch die - die aufgetretenen Probleme besprochen werden, die technischen Architekten treffen sich alle zwei Wochen. Ja, die Requirement-Engineers haben auch ihre Runde."* – T6, Technischer Architekt

Im Fallbeispiel 3 gibt es wöchentliche Meetings mit der Projektleitung, Release- und Test-Management, in denen offene Tickets priorisiert und bewertet werden. Zusätzlich stimmen sich Teamleiter regelmäßig mit Fachabteilungen ab, um technische und organisatorische Fragen zu klären. Diese Meetings sind essenziell, um Abhängigkeiten zwischen Teams frühzeitig zu erkennen und eine strukturierte Zusammenarbeit sicherzustellen.

Rollenspezifische Teamleiter-Meetings helfen dabei, den fachlichen Austausch gezielt zu organisieren und Wissen innerhalb der Teams strukturiert weiterzugeben. Während Dailies und Weeklies vor allem den aktuellen Arbeitsstand und teamübergreifende Abstimmungen behandeln, bieten diese Meetings eine Möglichkeit für detaillierte technische und methodische Diskussionen. So können Entscheidungen innerhalb einzelner Fachbereiche fundiert getroffen werden.
Die spezifischen Zielsetzungen und Abgrenzungen von Dailies, Weeklies und rollenspezifischen Teamleiter-Meetings sind in Tabelle 23 zusammengefasst.

Kriterium	6.1.7.1 Regelmäßige synchrone Abstimmung im Team	6.1.7.2 Wöchentliche Abstimmung zwischen Teams	6.1.7.3 Rollenspezifische Teamleiter-Meetings zur fachlichen Koordination
Fokus	Kurzfristige Abstimmung zum aktuellen Stand und Blockaden	Teamübergreifende Abstimmung zu laufenden Themen und Abhängigkeiten	Fachliche, technische oder organisatorische Abstimmung innerhalb spezifischer Rollen
Teilnehmer	Alle Teammitglieder	Vertreter mehrerer Teams, oft Entwickler, Tester, Requirements Engineers, teilweise Product Owner	Teamleiter, technische Architekten, Test- oder Fachbereichsverantwortliche
Frequenz	Täglich	Wöchentlich	Wöchentlich oder nach Bedarf
Scope	Innerhalb eines Teams	Zwischen mehreren Teams	Rollen- bzw. fachbereichsübergreifend
Typische Themen	Welche Aufgaben sind in Bearbeitung? Gibt es Hindernisse oder Klärungsbedarf?	Welche Schnittstellen oder Abhängigkeiten gibt es? Welche teamübergreifenden Probleme müssen gelöst werden?	Architekturentscheidungen, Teststrategien, Priorisierung von Aufgaben innerhalb eines Fachbereichs

Tabelle 23: Die verschiedenen Arten von Meetings

6.1.8 Fortschrittsmessung

Die Codes „Im Daily“, „wöchentliche Meetings mit Leitung“ und „Ticketstatus (%, Stunden oder Status)“ werden zu folgenden Best Practices ausformuliert:

6.1.8.1 Regelmäßige Fortschrittsbesprechungen

Während tägliche Abstimmungen (siehe Punkt 6.1.1.2) vorrangig der kurzfristigen Koordination innerhalb der Teams dienen, fokussieren sich die hier beschriebenen Fortschrittsbesprechungen auf eine übergreifende und strategische Bewertung des Projektfortschritts. Ziel ist es, Abweichungen vom Plan zu erkennen, Maßnahmen einzuleiten und eine fundierte Entscheidungsgrundlage für die weitere Projektsteuerung zu schaffen.

Im Fallbeispiel 1 gibt es wöchentliche Leistungsfortschrittsberichte, die im Rahmen von Besprechungen genutzt werden, um den aktuellen Status zu bewerten. Dabei wird eine prozentuale Einschätzung des Fortschritts vorgenommen, um größere Themenblöcke zu erfassen. Zusätzlich werden Abweichungen vom geplanten Verlauf besprochen und Maßnahmen zur Anpassung definiert.

Im Fallbeispiel 2 findet die Fortschrittskontrolle hauptsächlich über Sprint-Meetings statt. Hier werden abgeschlossene, laufende und anstehende Aufgaben besprochen, wobei Anforderungen regelmäßig überprüft und priorisiert werden. Tägliche Stand-up-Meetings (Dailies) ergänzen diesen Prozess, indem sie eine kurzfristige Fortschrittsbewertung innerhalb der Teams ermöglichen (siehe Punkt 6.1.1.2). Änderungen oder Verzögerungen werden in diesen Meetings transparent gemacht, sodass frühzeitig Gegenmaßnahmen eingeleitet werden können.
Im Fallbeispiel 3 erfolgt die Fortschrittsüberprüfung sowohl in wöchentlichen Meetings mit der Projektleitung als auch in teaminternen Abstimmungen. Neben der Diskussion offener Aufgaben liegt der Fokus auf der Bewertung technischer Implementierungen und potenzieller Risiken. In diesen Meetings werden zudem teamübergreifende Abhängigkeiten besprochen, um sicherzustellen, dass alle relevanten Stakeholder informiert sind und notwendige Entscheidungen zeitnah getroffen werden.

Besprechungen des Fortschritts werden in allen drei Fallbeispielen durch den Einsatz von Tools wie Jira oder Kanban-Boards unterstützt. Dort werden Statusinformationen bereitgestellt und eine Grundlage für die Diskussionen geschaffen. Im Unterschied zu täglichen Dailies oder reiner Tool-Transparenz dienen diese Meetings jedoch nicht nur der Statusmeldung, sondern ermöglichen

eine übergreifende, strukturierte Bewertung des Projektfortschritts. Sie bieten Raum, um Abweichungen vom Plan frühzeitig zu erkennen, Maßnahmen zur Kurskorrektur abzuleiten und die Zusammenarbeit im Team sowie mit der Projektleitung aktiv zu steuern.

6.1.8.2 Toolunterstützter Workflow

Die Best Practice „Toolunterstützter Workflow" wird bereits im Punkt 6.1.1.1 beschrieben und deckt auch einen Teil der Kategorie „Fortschrittsmessung" ab.

6.1.9 Auswahl & Zuteilung von Tasks

Die Codes „Leitung an Team" und „Auswahl durch Teammitglieder aus Task-Liste", „Zuteilung nach Fachwissen/jenen der sich am besten auskennt" werden zu folgenden Best Practices ausformuliert:

6.1.9.1 Koordinierten Aufgabenverteilung für kritische und spezialisierte Aufgaben

In autonomen Teams erfolgt die Vergabe der Aufgaben in der Regel flexibel. Es gibt jedoch bestimmte Situationen, in denen eine gezielte Zuweisung von Aufgaben sinnvoll ist, insbesondere bei sicherheitskritischen Themen, komplexen Anforderungen oder dringenden Fehlerbehebungen. In diesen Fällen übernimmt eine koordinierende Rolle, wie der Teamleiter oder ein Fachexperte, die Verantwortung, sicherzustellen, dass die Aufgaben an die am besten geeigneten Teammitglieder vergeben werden.

Im Fallbeispiel 1 gibt es eine weitgehend freie Aufgabenwahl, jedoch wird darauf geachtet, dass kritische Aufgaben gezielt zugewiesen werden. Der Teamleiter unterstützt dabei, den Überblick über anstehende Arbeiten zu behalten und bei Bedarf Aufgaben gezielt zu verteilen, um sicherzustellen, dass wichtige Meilensteine eingehalten werden.

Im Fallbeispiel 2 werden Anforderungen über Jira verwaltet und in Meetings priorisiert. Während reguläre Aufgaben von den Teammitgliedern selbst ausgewählt werden, gibt es Fälle, in denen Teamleiter oder Projektverantwortliche gezielt Aufgaben zuweisen, etwa wenn spezielle Expertise erforderlich ist oder ein dringender Bug behoben werden muss.

Im Fallbeispiel 3 erfolgt die Aufgabenvergabe hauptsächlich über Sprint-Meetings. Das Team entscheidet gemeinsam, wer welche Aufgaben übernimmt, jedoch gibt es bei hochkomplexen

oder sicherheitskritischen Anforderungen feste Vorgaben, welche Rollen für die Umsetzung verantwortlich sind.

> *„Die Zuteilung funktioniert dann so, dass […] die Teamleads diese Defects zugewiesen bekommen und dann für die weitere Verteilung in den Teams zuständig sind. Teilweise ist es aber auch so, wenn man weiß, dass eine bestimmte Person ganz sicher für das zuständig ist, dass die direkt diesen Defect zugeteilt bekommt. […] Die Auswahl erfolgt grundsätzlich danach, wer sich mit einem Thema auskennt oder wer gerade Zeit hat, sich etwas anzuschauen.“* – T4, Softwareentwickler und Tester

Diese Form der koordinierten Aufgabenverteilung hilft dabei, wichtige Aufgaben zuverlässig zu bearbeiten, ohne dass es durch unklare Zuständigkeiten zu Verzögerungen kommt. In allen drei Fallbeispielen zeigt sich als gemeinsame Praxis, dass Teamleiter oder fachlich zuständige Personen bei sicherheitskritischen, komplexen oder besonders dringenden Aufgaben gezielt eingreifen. So wird sichergestellt, dass die Aufgaben von den am besten geeigneten Teammitgliedern übernommen werden – ohne grundsätzlich die Selbstorganisation des Teams einzuschränken.

6.1.9.2 Förderung von Eigenverantwortung durch selbstbestimmte Aufgabenwahl

In autonomen Teams ist es wichtig, dass Teammitglieder selbst entscheiden können, welche Aufgaben sie übernehmen. Dies stärkt nicht nur die Eigenverantwortung, sondern sorgt auch für eine gleichmäßige Verteilung der Arbeit und ermöglicht es den Teammitgliedern, sich gemäß ihren Interessen und Stärken einzubringen. In den untersuchten Fallbeispielen erfolgt die Selbstzuweisung von Aufgaben über verschiedene Mechanismen, die sicherstellen, dass der Arbeitsfluss nicht gestört wird und alle wichtigen Aufgaben bearbeitet werden.
Im Fallbeispiel 1 wird die Aufgabenwahl weitgehend durch das Team selbst gesteuert. Teammitglieder entnehmen ihre Aufgaben aus einer Kanban-Liste oder einem Aufgaben-Board, wobei sie sich an ihrer aktuellen Verfügbarkeit oder ihrem Fachwissen orientieren. Die Zuteilung erfolgt oft informell oder wird in Meetings gemeinsam abgestimmt, um sicherzustellen, dass sich die Arbeitslast gleichmäßig verteilt.

Im Fallbeispiel 2 gibt es eine priorisierte Kanban-Liste, aus der sich Teammitglieder Aufgaben nehmen. Die Auswahl erfolgt individuell, wobei Defects und besonders kritische Anforderungen priorisiert und teilweise vom Product Owner oder Teamleiter gesteuert werden. Gleichzeitig wird darauf geachtet, dass sich Entwickler nicht nur auf ein Spezialgebiet konzentrieren, sondern auch unterschiedliche Aufgaben übernehmen, um Wissensinseln zu vermeiden.

> *„Als selbstorganisiertes Team haben wir ein Kanban-Board und jeder, der Arbeit braucht, der nimmt sich was von diesem Board."* – T8, Technischer Architekt und Softwareentwickler

Im Fallbeispiel 3 basiert die Selbstzuweisung der Aufgaben ebenfalls auf einer sichtbaren Aufgabenliste oder einem Board, wobei es keine festen Regeln gibt, wie diese übernommen werden. Manche Aufgaben werden in Meetings demokratisch verteilt, während sich Teammitglieder in anderen Fällen selbstständig Aufgaben zuweisen, die ihrer Expertise entsprechen oder gerade von niemand anderem bearbeitet werden. Besonders bei teamübergreifenden oder komplexen Themen stimmen sich die Beteiligten vorher ab, um sicherzustellen, dass alle relevanten Aspekte berücksichtigt werden.

> *„Das Team nimmt sich dann eben beim Planning aus dem Backlog die Tickets, sortiert nach Priorität und setzt diese dann um."* – T11, Teamleiter

Unabhängig von den individuellen Ausprägungen zeigt sich in allen drei Fallbeispielen ein gemeinsames Grundprinzip:

- Aufgaben sind für alle Teammitglieder sichtbar, entweder in einem Kanban-Board, einer To-do-Liste oder einem digitalen Tool wie Jira.
- Die Teammitglieder wählen Aufgaben eigenständig aus diesem Pool aus, wobei Fachwissen, Verfügbarkeit und Dringlichkeit eine Rolle spielen.
- Es gibt Mechanismen zur Abstimmung innerhalb der Teams, um sicherzustellen, dass keine Aufgaben übersehen werden und die Verteilung ausgeglichen bleibt.

Die selbstbestimmte Aufgabenwahl stellt sich in den untersuchten Fallbeispielen als bewährte Praxis dar, bei der Teammitglieder Aufgaben aus einem sichtbaren, priorisierten Aufgabenpool eigenständig auswählen. Diese Vorgehensweise fördert die Eigenverantwortung und erlaubt eine flexible Anpassung an individuelle Stärken und Verfügbarkeiten. Voraussetzung ist eine transparente Aufgabenübersicht – etwa in Form von Kanban-Boards – ergänzt durch regelmäßige Teamabsprachen, um eine gleichmäßige Verteilung sicherzustellen. Die Aufgaben werden mit Avataren oder Kürzeln gekennzeichnet, dadurch wird gekennzeichnet wer die Verantwortung für die Umsetzung übernommen hat. Im Unterschied zur koordinierten Aufgabenvergabe bei kritischen oder sicherheitsrelevanten Themen (siehe Punkt 6.1.9.1), bei der eine verantwortliche Person eine Erstzuweisung vornimmt, basiert diese Praktik auf freiwilliger Übernahme. Auch hier können Abstimmungen erfolgen, jedoch steht die eigenverantwortliche Wahl im Vordergrund.

6.1.10 Taskänderungen

Die Codes „Validierung und Einplanung nach Dringlichkeit“ und „Direkt, wenn nicht zu kritisch (aufgrund von Bürokratie)“ werden zu folgenden Best Practices ausformuliert:

6.1.10.1 Strukturierte Validierung und Priorisierung von Aufgabenänderungen

Fachliche und technische Änderungen sind fester Bestandteil der Entwicklungsarbeit und müssen effizient validiert, priorisiert und eingeplant werden. In allen drei Fallbeispielen wird bei jeder Änderung geprüft, ob sie noch im aktuellen Entwicklungszyklus berücksichtigt werden kann oder ob eine spätere Umsetzung sinnvoller ist. Kritische Anforderungen werden bevorzugt behandelt, während weniger dringliche Änderungen als Change Requests für spätere Releases dokumentiert werden.

Fachliche Änderungen werden in der Regel durch Product Owner oder Requirement Engineers angestoßen. Im Fallbeispiel 1 übernimmt der Teamleiter die Bewertung und entscheidet gemeinsam mit dem Product Owner, ob eine Änderung direkt umgesetzt werden kann oder auf einen späteren Zeitpunkt verschoben wird. Im Fallbeispiel 2 erfolgt die Priorisierung durch den Product Owner in Rücksprache mit dem Team. Der Requirement Engineer analysiert die Anforderungen und speist sie in das Backlog ein. Die Einplanung erfolgt abhängig von der Dringlichkeit und vom vorhandenen Kapazitätsrahmen. Im Fallbeispiel 3 ist aufgrund der gesetzlichen Vorgaben besonders wichtig, ob Änderungen noch innerhalb des vorgesehenen Release-Zyklus möglich sind. Bei kurzfristigen Anforderungen wird geprüft, ob eine Umsetzung rechtzeitig erfolgen kann, oder ob gemeinsam mit den Stakeholdern eine Nachlieferung vereinbart werden muss.

Technische Änderungen gehen häufig direkt von den Entwicklern aus, etwa bei Refactorings, Bugfixes oder Performance-Optimierungen. Im Fallbeispiel 1 werden diese zunächst im Team besprochen und bei Bedarf mit dem Teamleiter oder einem Architekturverantwortlichen abgestimmt. Bei umfangreicheren Änderungen wird zusätzlich das Software-Entscheidungsgremium (SEG) einbezogen. Im Fallbeispiel 2 liegt die Verantwortung für technische Bewertungen beim technischen Architekten. Änderungen, die Auswirkungen auf mehrere Komponenten haben, werden in Architektur-Meetings besprochen und abgestimmt. Im Fallbeispiel 3 erfolgt die technische Bewertung teamintern, wobei größere Änderungen mit der Projektleitung koordiniert und entsprechend der Release-Planung priorisiert werden.

> *„Wenn es jetzt natürlich auch andere Teams betrifft, muss man dort natürlich auch ein bisschen anfragen, und wenn es noch größer ist dann wird es bei dem technischen*

> *Architekten einmal Anklang finden […] Fachlich - wird das meist über die Requirement-Engineers gespielt. Dass die dann eine Entscheidung treffen können oder nicht - aber wenn es da auch noch einmal nach oben gehen muss dann liegt es am Product Owner ob das eine gültige Entscheidung ist oder nicht.“* – T10, Tester

> *„Wenn noch keine Entwicklung angefangen wurde, dann wird es halt eben adaptiert und neu geschätzt. […] Dann macht man einfach eine neue Story und eine neue Anforderung und man arbeitet additiv damit.“* – T9, Teamleiter und Requirements Engineer

In allen drei Fallbeispielen hat sich eine strukturierte Prüfung von Änderungsanforderungen etabliert, die fachliche und technische Aspekte gleichermaßen berücksichtigt. Die gemeinsame Best Practice besteht darin, jede Änderung anhand folgender Kriterien zu bewerten und einzuplanen:

- **Dringlichkeit**: Muss die Änderung sofort umgesetzt werden (z. B. aufgrund gesetzlicher Vorgaben oder kritischer Fehler)?
- **Machbarkeit**: Ist die Umsetzung im aktuellen Entwicklungszyklus realistisch und technisch möglich?
- **Abstimmung**: Ist eine Rücksprache mit dem Product Owner, dem Architekten oder der Projektleitung erforderlich?
- **Verantwortung**: Wer im Team verfügt über das nötige Wissen zur Umsetzung?
- **Einplanung oder Verschiebung**: Kann die Änderung direkt eingeplant werden, oder muss sie als Change Request dokumentiert und später berücksichtigt werden?

6.1.10.2 Schnelle Umsetzung unkritischer Änderungen zur Vermeidung bürokratischer Hürden

Nicht jede Änderung muss aufwendig validiert oder umfassend abgestimmt werden. In allen drei Fallbeispielen zeigt sich, dass kleinere Änderungen – wie einfache Bugfixes, UI-Korrekturen oder nicht-kritische Codeanpassungen – direkt durch die Entwickler umgesetzt werden können, sofern sie keine weitreichenden Auswirkungen auf andere Komponenten oder Teams haben.

Im Fallbeispiel 1 dürfen solche Änderungen ohne formale Freigabe umgesetzt werden, solange sie klar dokumentiert und technisch isoliert sind. Bei Unsicherheiten erfolgt eine kurze Abstimmung mit dem Teamleiter. Im Fallbeispiel 2 entnehmen Entwickler Aufgaben aus einem priorisierten Backlog oder Kanban-Board. Kleinere Änderungen werden eigenverantwortlich bearbeitet, während Änderungen mit potenziellen Seiteneffekten mit dem technischen Architekten oder

anderen Teams abgestimmt werden. Im Fallbeispiel 3 besteht eine definierte Grenze: Änderungen, die keine Auswirkungen auf Release-Termine oder regulatorische Vorgaben haben, können direkt umgesetzt werden, während größere Anpassungen über die Projektleitung koordiniert werden.

> *„Ja, es kommt natürlich auf den Umfang an, wie groß die Änderung ist. Wenn es eine Kleinigkeit ist, dann macht man es einfach…“* – T8, Technischer Architekt und Softwareentwickler

In der Praxis erfolgt die Umsetzung durch jene Person, die entweder die Änderung vorgeschlagen hat oder sich fachlich am besten mit dem betroffenen Bereich auskennt. Auch kleinere Änderungen werden über Tools wie Jira oder GitLab dokumentiert, um Nachvollziehbarkeit zu gewährleisten und sicherzustellen, dass andere Teammitglieder über den aktuellen Stand informiert bleiben.
Die gemeinsame Best Practice besteht darin, unkritische Änderungen gezielt und effizient umzusetzen, um die Eigenverantwortung im Team zu fördern und den Arbeitsfluss nicht zu verlangsamen. Vor der Umsetzung sollten folgende Punkte geprüft werden:

- **Komplexität**: Handelt es sich um eine isolierte, klar überschaubare Änderung ohne tiefere Abhängigkeiten?
- **Auswirkungen**: Gibt es technische oder organisatorische Seiteneffekte auf andere Komponenten oder Teams?
- **Abstimmung**: Reicht eine eigenverantwortliche Umsetzung, oder ist eine kurze Rücksprache mit Teamleitung oder Architekturverantwortlichen sinnvoll?
- **Dokumentation**: Ist sichergestellt, dass die Änderung im Ticket- oder Dokumentationssystem erfasst wird?
- **Zuständigkeit**: Übernimmt die Änderung die fachlich passendste Person im Team?

6.1.11 Team Events & Belohnungen

Der Code „Allgemeine Teamevents“ wird zu folgendem Best Practice ausformuliert:

6.1.11.1 Förderung des Teamzusammenhalts durch Teamevents

Regelmäßige gemeinsame Aktivitäten außerhalb des Arbeitskontexts tragen wesentlich zum sozialen Zusammenhalt in agilen Teams bei. Teamevents wie gemeinsames Essen, Spieleabende,

sportliche Ausflüge oder saisonale Feiern schaffen eine informelle Umgebung, in der sich Teammitglieder persönlich näherkommen und Beziehungen über die fachliche Zusammenarbeit hinaus aufbauen können.

In allen drei Fallbeispielen werden solche Aktivitäten als positiver Beitrag zur Teamkultur beschrieben. Im Fallbeispiel 1 werden Teamevents organisiert, um Austausch und Zusammenhalt zu stärken – etwa gemeinsame Abendessen oder Freizeitaktivitäten nach Projektphasen. Im Fallbeispiel 2 entstehen viele Aktivitäten aus dem Team heraus, darunter gemeinsame Mittagessen oder informelle Treffen, die den Zusammenhalt im Alltag fördern. Auch im Fallbeispiel 3 wird beschrieben, dass Teamevents gezielt eingesetzt werden, um die Motivation zu stärken und den respektvollen Umgang im Team zu fördern. Die Projektleitung unterstützt diese Initiativen und schafft bewusst Raum für informellen Austausch.

Solche Veranstaltungen erleichtern besonders neuen Teammitgliedern die Integration und helfen, Barrieren abzubauen. In einem ungezwungenen Rahmen fällt es leichter, Vertrauen aufzubauen, sich offen auszutauschen und gemeinsame Werte zu entwickeln – Aspekte, die sich positiv auf die spätere Zusammenarbeit im Projekt auswirken.

Teamevents können dabei nicht nur auf einzelne Entwicklungsteams begrenzt sein, sondern auch teamübergreifend organisiert werden – etwa für mehrere autonome Teams innerhalb eines größeren Projekts. Dies stärkt das Verständnis über Teamgrenzen hinweg und unterstützt den übergreifenden Zusammenhalt in der Organisation. Wichtig ist dabei, dass die Kosten solcher Veranstaltungen in der Regel vom Arbeitgeber getragen werden. Eine klare Kostenübernahme zeigt Wertschätzung, fördert die Beteiligung und vermeidet soziale Hürden für die Teilnahme.

In der Praxis haben sich folgende Ansätze bewährt:

- **Regelmäßigkeit**: Teamevents sollten fest in der Teamkultur verankert sein und über das Jahr verteilt stattfinden.
- **Vielfalt**: Unterschiedliche Formate – von kleinen spontanen Treffen bis hin zu größeren Ausflügen – berücksichtigen verschiedene Interessen.
- **Freiwilligkeit**: Die Teilnahme erfolgt ohne Erwartungsdruck, um eine lockere und authentische Atmosphäre zu gewährleisten.
- **Unterstützung durch das Management**: Eine positive Haltung der Projektleitung und die Übernahme der Kosten fördern Eigeninitiative und Beteiligung der Teams.

6.1.12 Komplexe Themen

Die Codes „4-Augen-Prinzip & Reviews“ und „Teamabsprache und Validierung“ werden zu folgenden Best Practices ausformuliert:

6.1.12.1 Pair-Programming, 4-Augen-Prinzip und Reviews

Die Best Practice „Pair-Programming, 4-Augen-Prinzip und Reviews“ wird bereits im Punkt 6.1.3.2 beschrieben und deckt auch einen Teil der Kategorie „Komplexe Themen“ ab.

6.1.12.2 Strukturierte Teamabstimmung und Validierung bei komplexen Aufgaben

Bei komplexen Aufgaben greifen die Teams in den Fallbeispielen auf gezielte Abstimmungen zurück. Ziel ist es, fachliche und technische Fragen frühzeitig zu klären, Risiken zu minimieren und tragfähige Lösungen zu entwickeln. In allen drei Fallbeispielen zeigt sich ein gemeinsames Vorgehen, auch wenn es in der Ausprägung unterschiedlich ist.

Im Fallbeispiel 1 landen komplexere Themen meist zuerst beim Teamleiter oder einer erfahrenen Person im Team. Es erfolgt eine erste Einschätzung, ob die Aufgabe intern geklärt werden kann oder ob weitere Abstimmungen nötig sind. Bei Bedarf werden eigene Runden einberufen, in denen mehrere Personen ihre Perspektiven einbringen. Dabei wird Wert daraufgelegt, die Umsetzung nicht vorschnell zu starten, sondern vorab relevante Fragen gemeinsam zu klären.

Im Fallbeispiel 2 zeigt sich ein ähnlicher Ablauf. Themen mit größerer Tragweite werden oft zunächst in kleiner Runde vorbereitet – etwa in Fokusgruppen oder technischen Abstimmungen – bevor sie im gesamten Team besprochen und entschieden werden. Besonders bei teamübergreifenden oder architekturrelevanten Themen werden technische Ansprechpartner und Architekten frühzeitig eingebunden.

Im Fallbeispiel 3 ist dieses Vorgehen besonders strukturiert. Komplexe Themen werden vorab von erfahrenen Entwicklern vorbereitet und in übersichtlicher Form aufbereitet. Das kann Skizzen, erste Lösungsentwürfe oder eine strukturierte Analyse beinhalten. Anschließend wird das Thema dem Team vorgestellt und gemeinsam diskutiert. Wenn intern nicht genug Know-how vorhanden ist, wird gezielt externe Expertise eingebunden. Der Umgang mit komplexen Themen wird dabei als bewusst gemeinsamer Prozess verstanden – man bringt die richtigen Leute zusammen, prüft verschiedene Optionen und trifft dann die bestmögliche Entscheidung.

Insgesamt zeigt sich, dass solche Themen nicht isoliert bearbeitet werden. Stattdessen wird im Vorfeld geklärt, wer eingebunden werden muss, wie die Entscheidung vorbereitet wird und wer die Umsetzung übernimmt. In der Praxis haben sich folgende Punkte bewährt:

- Komplexe Themen werden gezielt vorbereitet, meist durch erfahrene Personen oder kleinere Gruppen.
- Fachliche und technische Fragen werden im Team strukturiert besprochen.
- Bei Bedarf werden andere Rollen wie Architekten, Product Owner oder externe Experten eingebunden.
- Die Umsetzung übernimmt die Person, die fachlich am besten zum Thema passt.
- Entscheidungen werden gemeinsam getroffen und nachvollziehbar dokumentiert.

6.1.13 Aufwandsschätzung

Die Codes „Schätzung über mehrere Iterationen“, „Durch Teammitglieder“, „Aufteilung in Subtasks“ und „Projektspezifische Einheitenschätzung“ werden zu folgenden Best Practices ausformuliert:

6.1.13.1 Iterative Verfeinerung der Schätzungen durch das gesamte Team

In allen drei Fallbeispielen werden Aufwände nicht einmalig geschätzt, sondern über den Projektverlauf hinweg iterativ verfeinert. Ziel ist es, durch gewonnene Erfahrungswerte die Planungsgenauigkeit schrittweise zu verbessern und realistischere Prognosen zu ermöglichen.

Im Fallbeispiel 1 erfolgt die Aufwandsschätzung pragmatisch und basiert meist auf Schätzungen in Personentagen. Die Einschätzungen werden gemeinsam im Team vorgenommen – in der Regel von den Teammitgliedern selbst, wobei der Teamleiter die Verantwortung für die Nachvollziehbarkeit und Korrektheit übernimmt. Bei spezifischen Aufgaben kann auch in kleineren Gruppen geschätzt werden.

Im Fallbeispiel 2 finden Estimation-Meetings statt, in denen technische Experten wie Requirements Engineers oder Architekten gemeinsam mit den Entwicklern Aufwände abschätzen. Je nach Thema wird der Prozess durch kleinere Fokusgruppen unterstützt – etwa für bestimmte Technologien oder Komponenten. Die Schätzwerte basieren auf Erfahrungswerten und werden im Projektverlauf bei Bedarf überarbeitet, um die Genauigkeit zu erhöhen.

Im Fallbeispiel 3 kommt Planning Poker zum Einsatz. Die Teammitglieder geben unabhängig voneinander ihre Einschätzung in Story Points ab. Bei größeren Abweichungen wird im Team diskutiert, bis ein Konsens gefunden ist. Durch diesen Austausch, aber auch durch den Vergleich mit abgeschlossenen Stories, verbessert sich die Genauigkeit im Laufe der Zeit. Zusätzlich ergibt sich daraus ein klareres Bild zur Team-Velocity – also der Anzahl an Story Points, die pro Iteration realistisch umgesetzt werden können.

> *„… die Teams und die Teamleader diese Aufwandsschätzungen durchführen und auch mit den einzelnen Entwicklern genauer abstimmen, wie viel Aufwand das bedeutet oder was eigentlich zu tun ist. […] grundsätzlich wird das schon - von also aus Entwicklungssicht von den zuständigen Entwicklern oder deren zuständigen Lead gemacht. […] sodass in Summe dann halt eine gemeinsame Schätzung rauskommt.“* – T4, Softwareentwickler und Tester

> *„Die Estimations machen wir im Team. […] Wir haben unsere Storys im Backlog und machen einfach Planning Poker. […] Wir schätzen die Komplexität.“* – T11, Teamleiter

Die gemeinsame Best Practice lässt sich wie folgt zusammenfassen:

- Aufwände werden gemeinsam im Team geschätzt – je nach Organisationsvorgabe in Stunden, oder Story Points.
- Die Schätzungen basieren auf Erfahrungswerten und werden regelmäßig nach Iterationen oder neuen Erkenntnissen angepasst.
- Bei komplexen Themen übernehmen kleinere Fokusgruppen mit spezifischem Fachwissen die Einschätzung.
- Methoden wie Planning Poker fördern Diskussion, Perspektivenabgleich und gemeinsame Lernprozesse.
- Die iterative Verfeinerung der Werte erhöht die Planungsgenauigkeit im Projektverlauf.

6.1.13.2 Aufteilung in Subtasks zur besseren Planung

Die strukturierte Aufteilung größerer Aufgaben in kleinere Einheiten ist ein bewährter Bestandteil der Aufwandsschätzung und Planung. In allen drei Fallbeispielen wird diese Vorgehensweise angewendet, um Schätzungen zu erleichtern, die Umsetzung klarer zu machen und den Fortschritt besser nachverfolgen zu können.

Im Fallbeispiel 1 werden größere Aufgaben in Teilaufgaben zerlegt, bevor sie geschätzt und eingeplant werden. Die Aufteilung erfolgt mit dem Ziel, besser planen und kontrollieren zu können, ob der Umfang realistisch ist. Diese Vorgehensweise wird im Team durchgeführt und hilft dabei, eine bessere Einschätzung für einzelne Entwicklungsschritte zu erhalten.

Im Fallbeispiel 2 werden Anforderungen vom Requirements Engineering Team so vorbereitet, dass sie in kleinere Bestandteile aufgeteilt und dann geschätzt werden können. Die Zerlegung erfolgt vor der eigentlichen Aufwandsschätzung und unterstützt die Diskussion und Abgrenzung der Themen im Team. Dabei wird betont, dass eine gute Vorbereitung notwendig ist, um eine sinnvolle Einschätzung durch die Teammitglieder zu ermöglichen.

Im Fallbeispiel 3 werden größere Stories regelmäßig in kleinere Aufgaben zerlegt. Diese Aufteilung erfolgt entweder direkt durch das Team oder wird durch das Requirements Engineering vorbereitet. Die aufgeteilten Bestandteile werden dann im Estimation-Meeting besprochen und geschätzt. Es wird bewusst darauf geachtet, dass Anforderungen erst dann geschätzt werden, wenn sie ausreichend verfeinert und verständlich sind.

Die gemeinsame Best Practice aus den Fallbeispielen lässt sich wie folgt zusammenfassen:

- Größere Aufgaben oder Stories werden vor der Schätzung in kleinere Bestandteile zerlegt.
- Die Vorbereitung wird durch Teammitglieder vorgenommen, erfolgt jedoch in enger Abstimmung mit dem Requirements Engineering.
- Die aufgeteilten Bestandteile werden im Team diskutiert und gemeinsam geschätzt.
  Die Zerlegung hilft, Missverständnisse zu vermeiden und die Aufgaben klarer zu strukturieren.

Nr.	Kapitel	Best Practice	Kategorie	Kategoriespezifische Umsetzung	Kurzbeschreibung
1	6.1.1.1	Toolunterstützter Workflow	Transparenz	Nutzung digitaler Kanban-Boards (z. B. Jira, GitLab) mit Status wie „In Bearbeitung“ oder „Done“.	Die Tickets werden in digitalen Tools wie Jira oder GitLab auf Boards verwaltet, um den Bearbeitungsstand über klar definierte Status transparent darzustellen. Der gesamte Workflow bleibt nachvollziehbar und zugänglich.
2	6.1.2.2		Kommunikation	Kommentarfunktionen in Jira/GitLab, Verknüpfung mit Meetings.	In Tools wie Jira oder GitLab können Teammitglieder Kommentare zu Tickets hinterlassen, um aktuelle Probleme, offene Fragen oder wichtige Abspra-

					chen zu dokumentieren und direkt im Kontext zu kommunizieren.
3	6.1.4.2		Wissensverteilung & Support	Regelmäßige Dokumentation in Wikis (Inhalte, Rollen, Standards, Entscheidungen)	Confluence oder Wikis ermöglichen das gezielte Nachlesen von technischen Dokumentationen, API-Beschreibungen oder Ansprechpersonen – hilfreich bei Supportfällen oder zur Einarbeitung neuer Mitglieder.
4	6.1.8.2		Fortschrittsmessung	Sprintfortschritt über Dashboards und Zeiterfassung in Tools wie Jira.	Fortschrittsanzeigen in Dashboards, Boards oder über Zeiterfassung erlauben eine objektive Statusbewertung des Arbeitsfortschritts innerhalb eines Sprints oder Releases. So lassen sich frühzeitig Abweichungen erkennen.
5	6.1.1.2	Regelmäßige synchrone Abstimmung im Team	Transparenz	Dailies mit kurzer Statusmeldung, Abgleich von Aufgaben und Blockaden.	Tägliche, meist 15-minütige Dailies ermöglichen den schnellen Abgleich des Arbeitsstands, die Erkennung von Blockern und die gemeinsame Sicht auf den Projektstatus. Zudem ist ersichtlich, welche Person welche Aufgabe bearbeitet.
6	6.1.2.3		Kommunikation	Kurze tägliche Teammeetings für gegenseitigen Abgleich.	Im Daily berichten Teammitglieder in kurzen Runden über Fortschritt und Probleme. So wird gegenseitiges Verständnis gefördert und Informationslücken werden minimiert.
7	6.1.7.1		Meetings	Tägliche Dailies, meist als 15-minütige Kurzmeetings, zur Abstimmung des Arbeitsstands und zur Identifikation von Blockern direkt am Kanban- oder Sprint-Board.	Dailies dienen als fixes, kurzes Format zur Abstimmung des aktuellen Stands im Team. Sie unterstützen Transparenz und regelmäßige Synchronisation bei laufender Arbeit.
8	6.1.1.3	Wöchentliche Abstimmung zwischen Teams	Transparenz	Wöchentliche Abstimmung mit Vertretern mehrerer Teams zu Abhängigkeiten, Problemen und Release-Planung. Dokumentation erfolgt z. B. in Confluence.	Ermöglicht teamübergreifende Transparenz über Fortschritte, Abhängigkeiten und Probleme. Teilnehmende Teams erhalten ein gemeinsames Bild über offene Themen und geplante Umsetzungen.
9	6.1.7.2		Meetings	Koordination technischer	Regelmäßige strukturierte Koordinationsrunden zwischen Teams

				Schnittstellen, Priorisierung von Tickets und Beteiligung von Architektur-, Test- und Projektverantwortlichen im Weekly-Format.	zur Abstimmung technischer Schnittstellen, Priorisierung von Aufgaben und Release-Management. Entscheidungen und Zuständigkeiten werden dokumentiert.
10	6.1.1.4	Standardisierte Protokollierung und Dokumentation	Transparenz	Nutzung strukturierter Vorlagen und der Definition of Done/Ready zur Nachvollziehbarkeit.	Dokumentation von Entscheidungen, Reviews und Aufgabenstandards (z. B. Definition of Done) sichert Wissen im Team, erleichtert Einarbeitung und schafft Verbindlichkeit für die Umsetzung.
11	6.1.2.1	Effiziente Abstimmung durch direkte Kommunikation	Kommunikation	Verwendung von Face-to-Face, Chat, Telefon und Screen-Sharing.	Durch direkte Gespräche, Chat oder Anrufe kann schnell und flexibel auf Rückfragen oder Probleme reagiert werden – ohne den Umweg über formelle Dokumentation.
12	6.1.4.3		Wissensverteilung & Support	Direkter Austausch bei Rückfragen oder Einarbeitung neuer Kollegen.	Die Praxis fördert unmittelbaren Wissenstransfer.
13	6.1.3.1	Regelmäßige Reflexion und Feedbackschleifen	Reflexion & Lernprozess	Sprint-Retrospektiven zur Reflexion	Sprint-Retrospektiven zur Reflexion von Arbeitsweisen sowie ergänzende technische Reviews zur kontinuierlichen Verbesserung der Codequalität. Erkenntnisse werden dokumentiert und fließen in zukünftige Iterationen ein.
14	6.1.3.2	Pair-Programming, 4-Augen-Prinzip und Reviews	Reflexion & Lernprozess	Review-Formate wie das 4-Augen-Prinzip bieten strukturierte Möglichkeiten zur Reflexion über die Qualität und Angemessenheit von Lösungen und fördern den kontinuierlichen Lernprozess im Team.	Regelmäßige Code-Reviews fördern konstruktives Feedback, ermöglichen eine kritische Reflexion der Codequalität und stärken die gemeinsame Verantwortung im Team.
15	6.1.4.1		Wissensverteilung & Support	Pair-Programming und Reviews fördern den Wissenstransfer und helfen neuen	Pair-Programming und strukturierte Reviews erleichtern die Einarbeitung, fördern kontinuierlichen Wissenstransfer und verhindern Wissensinseln im Team.

				Teammitgliedern bei der schnellen Einarbeitung.	
16	6.1.12.1		Komplexe Themen	Komplexe Themen werden gemeinsam analysiert und gelöst – etwa durch gemeinsames Debugging, strukturierte Abstimmungen oder Pair-Programming, wodurch tragfähige Lösungen entstehen.	Bei besonders anspruchsvollen Aufgaben werden durch gemeinsames Debugging, fachliche Abstimmungen oder Pair-Programming fundierte Lösungen gemeinsam erarbeitet.
17	6.1.4.4	Mehrere Themenverantwortliche zur Wissensverteilung	Wissensverteilung & Support	Jedem Thema werden mindestens zwei Personen zugewiesen	Für jedes zentrale Thema sind bewusst mehrere Teammitglieder verantwortlich, um Wissensinseln zu vermeiden und die Ausfallssicherheit sowie Flexibilität bei der Bearbeitung zu erhöhen.
18	6.1.5.1	Weiterbildung durch Eigeninitiative und Teamförderung	Fortbildung	Zeitfenster für Weiterbildung auf Eigeninitiative. Teils mit Zertifikaten oder Lernaufgaben.	Teammitglieder stoßen Fortbildung eigenständig an. Unterstützung erfolgt durch Teamleiter z. B. über Lernzeit, Zertifikatskosten oder individuelle Technologieerprobung. Lernen erfolgt projektintegriert und praxisnah.
19	6.1.5.2	Themenwechsel zur individuellen Weiterentwicklung	Fortbildung	Entwickler wechseln auf Initiative in neue Fachbereiche, oft schrittweise durch kleinere Aufgaben.	Teammitglieder wechseln eigeninitiativ in neue Themenbereiche, um Wissen zu verbreitern und neue Fähigkeiten aufzubauen. Wechsel erfolgen schrittweise und in Abstimmung mit dem Team.
20	6.1.5.3	Rollenwechsel zur Entwicklung individueller Fähigkeiten	Fortbildung	Ein schrittweiser Rollenwechsel, z. B. vom Entwickler zum Tester oder Requirements Engineer.	Der Wechsel in andere fachliche Rollen – z. B. von Entwicklung zu Test – ermöglicht flexible Teamaufstellung und individuelle Weiterentwicklung. Die Übergabe erfolgt meist schrittweise.
21	6.1.6.1	Dokumentation und Festlegung von Richtlinien und Prozessen	Richtlinien, Grenzen, Normen und Prinzipien	Globale Richtlinien werden durch Architekten oder Gremien erstellt und dokumentiert.	Verbindliche technische Vorgaben wie Architekturprinzipien und Coding-Standards werden systemweit definiert und in Wikis dokumentiert. Die Umsetzung wird von technischen Architekten und Teamleitern begleitet, um Konsistenz über alle Teams hinweg sicherzustellen.

22	6.1.6.2	Teamspezifische Erweiterungen von Qualitätsstandards	Richtlinien, Grenzen, Normen und Prinzipien	Zusätzliche Teamrichtlinien wie Checklisten, Vorgaben zur Testabdeckung oder Definition of Done über das Standardniveau hinaus.	Teams ergänzen zentrale Richtlinien durch eigene Qualitätsstandards wie Checklisten oder Definition of Done. Tools wie SonarQube oder Checkstyle unterstützen die automatisierte Einhaltung.
23	6.1.7.3	Rollenspezifische Teamleiter-Meetings zur fachlichen Koordination	Meetings	Architekten, Tester, etc. tauschen sich regelmäßig in Rollenrunden aus.	Rollenspezifische Teamleiter-Meetings ermöglichen fachliche, technische oder organisatorische Abstimmungen über Teamgrenzen hinweg und unterstützen z. B. Architektur- oder Testentscheidungen.
24	6.1.8.1	Regelmäßige Fortschrittsbesprechungen	Fortschrittsmessung	Wöchentliche Statusmeetings z.B. mit der Projektleitung	Wöchentliche Meetings der Teamleader mit der Projektleitung zur strukturierten Bewertung des Projektfortschritts, um Abweichungen frühzeitig zu erkennen und Maßnahmen zu koordinieren.
25	6.1.9.1	Koordinierte Aufgabenverteilung für kritische und spezialisierte Aufgaben	Auswahl & Zuteilung von Tasks	Teamleiter weisen sicherheitskritische oder komplexe Aufgaben gezielt zu.	Bei kritischen Aufgaben erfolgt eine Erstzuweisung durch Teamleitung oder Fachverantwortliche an die geeignetsten Personen, wobei die grundsätzliche Selbstorganisation erhalten bleibt.
26	6.1.9.2	Förderung von Eigenverantwortung durch selbstbestimmte Aufgabenwahl	Auswahl & Zuteilung von Tasks	Aufgabenwahl aus Kanban-Board nach Interesse, Wissen und Kapazität.	Teammitglieder wählen eigenständig Aufgaben aus einem priorisierten Aufgabenpool oder digitalen Board aus, wodurch Motivation, Verantwortungsbewusstsein und fachliche Passung gestärkt werden.
27	6.1.10.1	Strukturierte Validierung und Priorisierung von Aufgabenänderungen	Task-Änderungen	Abstimmung mit PO oder Architekt je nach Dringlichkeit und Aufwand.	Kleine technische Änderungen wie Bugfixes, UI-Verbesserungen oder Typos können direkt umgesetzt werden, sofern sie isoliert sind und keine Abstimmung mit anderen Teams erfordern. Dies beschleunigt den Workflow und reduziert bürokratischen Aufwand.
28	6.1.10.2	Schnelle Umsetzung unkritischer Änderungen zur Vermeidung bürokratischer Hürden	Task-Änderungen	Kleine Änderungen direkt durch Entwickler, dokumentiert im Ticket-Tool.	Kleine technische Änderungen wie Bugfixes können direkt umgesetzt werden, sofern sie isoliert sind und keine Abstimmung mit anderen Teams erfordern. Dies beschleunigt den Workflow und reduziert bürokratischen Aufwand.
29	6.1.11.1	Förderung des Teamzusammenhalts durch Teamevents	Team Events & Belohnungen	Regelmäßige Events (Mittagessen, Spieleabende),	Gemeinsame Teamevents wie Ausflüge, Feiern oder Mittagessen stärken Vertrauen und Motivation. Die Events erleichtern

				teilweise team-übergreifend.	neue Kontakte, fördern Integration und Teambindung.
30	6.1.12.2	Strukturierte Teamabstimmung und Validierung bei komplexen Aufgaben	Komplexe Themen	Vorabklärung in kleinen Gruppen, oft mit Skizzen oder Voranalysen.	Bei komplexen Aufgaben stimmen sich erfahrene Teammitglieder gezielt ab. Technische Entwürfe werden vorbereitet und gemeinsam diskutiert. Falls erforderlich, werden externe Experten eingebunden.
31	6.1.13.1	Iterative Verfeinerung der Schätzungen durch das gesamte Team	Aufwandsschätzung	Planning Poker oder Meetings mit Team zur Aufwandsschätzung und Justierung pro Sprint.	Aufwände werden im Team gemeinsam geschätzt – z. B. über Planning Poker. Bei größeren Abweichungen wird diskutiert, bis ein Konsens entsteht. Die Schätzungen werden nach Iterationen überprüft und verfeinert.
32	6.1.13.2	Aufteilung in Subtasks zur besseren Planung	Aufwandsschätzung	Große Tasks werden vor dem Sprint zerlegt und einzeln geschätzt.	Komplexere Aufgaben werden in kleinere, besser plan- und schätzbare Einheiten aufgeteilt, um Missverständnisse zu vermeiden und eine detaillierte Umsetzungsvorbereitung zu ermöglichen.

Tabelle 24: Auflistung aller gefundenen Best Practices

6.2 Analyse skalierbarer Best Practices

In diesem Kapitel wird untersucht, welche der identifizierten Best Practices aus den drei Fallbeispielen Gemeinsamkeiten mit jenen von Hoda et al. aufweisen und somit skalierbar sind. Skalierbar im Sinne dieser Arbeit bedeutet, dass eine Praktik, die in kleinen, selbstorganisierten Teams erfolgreich angewendet wird, auch in größeren, komplexeren Projektkontexten mit mehreren Teams, Rollen und Koordinierungsebenen nachweislich beobachtet und wirksam eingesetzt werden kann. Die Analyse der Skalierbarkeit ermöglicht es, etablierte Prinzipien aus kleineren agilen Teams gezielt auf groß angelegte Softwareprojekte zu übertragen und liefert damit evidenzbasierte Handlungsempfehlungen für die Organisation komplexer Entwicklungsstrukturen.

Die Bewertung der Skalierbarkeit erfolgt auf Grundlage eines systematischen inhaltlichen Vergleichs: Dabei werden konkrete Praktiken aus den Fallstudien (siehe Kapitel 6.1) mit den von Hoda et al. beschriebenen Praktiken (siehe Kapitel 3.2) abgeglichen. Als Übereinstimmung gilt eine inhaltliche Entsprechung der Zielsetzung, des praktischen Vorgehens und des Kontexts der Anwendung. Grundlage sind kodierte Interviewsegmente, aus denen in Kapitel 6.1 Best Practices abgeleitet wurden. Eine Übereinstimmung wird dann als Beleg für Skalierbarkeit gewertet, wenn sich zeigt, dass dieselbe Praktik auch unter den komplexeren Bedingungen eines Softwaregroßprojekts mit vergleichbarem Zweck und Wirkung umgesetzt wird.

Ziel ist die Beantwortung der Forschungsfrage RQ1b: „Welche Praktiken für autonome Teams sind auch für große Softwareprojekte skalierbar?“ Insgesamt wurden die 13 Praktiken aus sieben Kategorien von Hoda et al. analysiert. Neun dieser Praktiken zeigen inhaltliche Entsprechungen zu Praktiken in den Fallbeispielen und werden in den Abschnitten 6.2.1 bis 6.2.9 beschrieben. Kapitel 6.2.10 behandelt die Praktiken, bei denen keine Übereinstimmung festgestellt werden konnte. Eine vollständige Übersicht mit den Zuordnungen zu Kategorien und Skalierbarkeit findet sich in Tabelle 25 am Ende dieses Kapitels.

6.2.1 Übereinstimmendes Practice 1: Collective Estimation & Planning

Diese Praktik gehört zu der Kategorie „Collective Decision-Making“ (siehe Kapitel 3.2.1) von Hoda et al. Diese Praktik beschreibt den Prozess, bei dem Teams Aufgaben gemeinsam schätzen und Iterationen planen. Schätzungen werden im Team durchgeführt, um ein gemeinsames Verständnis der Aufgaben und ihrer Komplexität zu erreichen. Ein Beispiel für ein solches Practice ist die Methode Planning Poker, die sowohl in Hodas Forschung als auch in den Fallbeispielen explizit erwähnt wird. Planning Poker erlaubt es den Teammitgliedern, die Komplexität von Aufgaben zu bewerten, Diskrepanzen in den Einschätzungen aufzudecken und diese durch Diskussionen aufzulösen.

Übereinstimmung mit: „Iterative Verfeinerung der Schätzungen durch das gesamte Team“ (siehe Punkt 6.1.13.1).

Begründung: Teambasierte Schätzungen sind in beiden Kontexten ein zentrales Element. Planning Poker wird dabei als Beispiel einer spezifischen Praktik hervorgehoben, welche sowohl Transparenz als auch ein gemeinsames Verständnis in der Bewertung von Aufgaben fördert. Teams können durch diesen Ansatz realistische Schätzungen abgeben, um Aufgaben einzuplanen.

6.2.2 Übereinstimmendes Practice 2: Daily Standup Meetings

Diese Praktik gehört zur Kategorie Self-Monitoring (siehe Kapitel 3.2.3) von Hoda et al. Daily Standup Meetings bieten eine Plattform, um Fortschritte, Hindernisse und die Tagesplanung abzustimmen. Diese Meetings finden täglich statt und ermöglichen es, alle Teammitglieder auf den gleichen Stand zu bringen. Jedes Teammitglied berichtet, was es am Vortag erreicht hat, was für den aktuellen Tag geplant ist und ob es Hindernisse gibt.

Übereinstimmung mit: „Regelmäßige synchrone Abstimmung im Team“ (siehe Punkt 6.1.1.2).

Begründung: Daily Standups sind in beiden Kontexten ein essenzielles Werkzeug, um die Transparenz und Synchronisation im Team zu gewährleisten. Durch ihre regelmäßige und strukturierte Durchführung wird sichergestellt, dass alle Teammitglieder auf dem gleichen Stand sind und Probleme frühzeitig identifiziert werden können. Diese Meetings fördern die interne Abstimmung und dienen als Grundlage für weiterführende Diskussionen, die gegebenenfalls in separaten Meetings vertieft werden.

6.2.3 Übereinstimmendes Practice 3: Information Radiators

Diese Praktik aus der Kategorie „Self-Monitoring“ (siehe Kapitel 3.2.3) beschreibt die Visualisierung des Projektfortschritts für alle sichtbaren Artefakte. Information Radiators – etwa Kanban-Boards oder Storyboards – dienen nicht nur der Selbstüberwachung des Teams, sondern auch der teamweiten Synchronisation. In der Forschung von Hoda et al. wird beschrieben, wie Teams Spalten wie „Not Assigned“, „Check-Out“, und „Done“ verwenden, um den Fortschritt zu strukturieren. Diese Spalten zeigen auch, wer an welchen Aufgaben arbeitet, häufig gekennzeichnet durch Avatare oder Initialen. Burndown-Charts ergänzen die Visualisierung, indem sie den verbleibenden Arbeitsaufwand auf Iterationsebene darstellen.

In den drei Fallbeispielen kommen ähnliche Tools wie z.B. Jira zum Einsatz, die Kanban-Boards und Fortschrittscharts bieten. Diese helfen, Aufgaben zu verfolgen und Engpässe frühzeitig zu erkennen. Die visuelle Darstellung unterstützt sowohl die Transparenz als auch die kontinuierliche Nachvollziehbarkeit innerhalb des Teams.

Übereinstimmung mit: „Toolunterstützter Workflow“ (siehe Punkt 6.1.1.1).

Begründung: Die Übereinstimmung ergibt sich aus der gemeinsamen Zielsetzung, Transparenz über den Fortschritt und die Verantwortlichkeiten im Team zu schaffen. Sowohl Hoda et al. als auch die Fallbeispiele betonen die Bedeutung visueller Werkzeuge zur Unterstützung der Zusammenarbeit und der kontinuierlichen Fortschrittsüberwachung.

6.2.4 Übereinstimmendes Practice 4: Using Story Board

Diese Praktik gehört zur Kategorie „Self-Assignment“ (siehe Kapitel 3.2.2) von Hoda et al. Storyboards, auch bekannt als Scrum-Boards, strukturieren die Aufgabenverteilung innerhalb eines Teams visuell und fördern die Selbstzuweisung und Priorisierung. Sie enthalten User Stories und deren technische Unteraufgaben, die das Team in einer Iteration umsetzen möchte. Diese Aufgaben werden auf kleinen Zetteln oder Post-its dargestellt und typischerweise in drei Spalten – „Not

Assigned“, „Check-Out“ und „Done“ – organisiert. Die Spaltenstruktur unterstützt nicht nur die Visualisierung des Bearbeitungsstandes, sondern erleichtert auch die teaminterne Koordination von Aufgaben und Zuständigkeiten.

In den Fallbeispielen wird eine ähnliche Nutzung von Storyboards beschrieben. Digitale Tools wie z.B. Jira dienen als zentrale Plattformen, die neben der Visualisierung von Aufgaben auch Funktionen zur Fortschrittsmessung bieten. Sie ermöglichen die automatische Verknüpfung von Aufgabenstatus mit Verantwortlichkeiten und unterstützen so die Nachverfolgbarkeit.

Übereinstimmung mit: „Toolunterstützter Workflow“ (siehe Punkt 6.1.1.1).

Begründung: In beiden Kontexten wird betont, wie wichtig die Sichtbarkeit des Arbeitsfortschritts für die Teamkoordination ist. Storyboards und digitale Tools fördern Transparenz und erleichtern die Nachverfolgung von Aufgaben.

6.2.5 Übereinstimmendes Practice 5: Taking Task Ownership

Ebenfalls aus der Kategorie „Self-Assignment“ (siehe Kapitel 3.2.2) beschreibt diese Praktik, wie Teammitglieder Aufgaben eigenständig übernehmen und so aktiv Verantwortung für deren Umsetzung tragen. In der Forschung von Hoda et al. wird beschrieben, wie Teammitglieder Aufgaben am Storyboard übernehmen, indem sie Post-its von „Not Assigned“ nach „Check-Out“ verschieben und damit die Verantwortung für die Umsetzung signalisieren. Diese Selbstzuweisung wird oft durch Initialen oder Avatare sichtbar gemacht. Darüber hinaus zeigt Hoda, dass die Motivation zur Aufgabenübernahme nicht in der Einfachheit der Aufgabe liegt, sondern im angestrebten geschäftlichen Nutzen (Engl. „Business Value“). Daraus ergibt sich häufig, dass Teammitglieder Aufgaben außerhalb ihres gewohnten Fachgebiets übernehmen – was gezielt zur Förderung von Cross-Funktionalität beiträgt.

In den Fallbeispielen erfolgt die Selbstzuweisung ebenfalls durch das Team, wobei die Auswahl auf Basis von Priorität, Verfügbarkeit und individuellem Interesse erfolgt. Dabei wird darauf geachtet, dass sich Teammitglieder nicht ausschließlich auf ihr Spezialgebiet konzentrieren, sondern gezielt auch fachfremde Aufgaben übernehmen, um Wissensinseln zu vermeiden und Weiterentwicklung zu ermöglichen. Die Aufgaben sind für alle sichtbar, meist über ein Kanban-Board oder eine priorisierte Liste. Die Zuweisungen erfolgen über Avatare oder Initialen. Die Teams achten bewusst auf eine sinnvolle Aufgabenverteilung, oft auch mit kollegialer Abstimmung, um Überlastung zu vermeiden und Lernchancen zu schaffen.

Übereinstimmung mit: „Förderung von Eigenverantwortung durch selbstbestimmte Aufgabenwahl“ (siehe Punkt 6.1.9.2).

Begründung: Die Teams übernehmen bewusst Aufgaben mit hohem Nutzen für das Projekt – auch außerhalb der eigenen Komfortzone. Damit wird nicht nur Eigenverantwortung gestärkt, sondern auch die funktionale Breite innerhalb des Teams gefördert. Die Übernahme von Aufgaben erfolgt nicht zufällig oder automatisiert, sondern ist das Ergebnis eines aktiven Entscheidungsprozesses, der Eigenverantwortung, Projektprioritäten und persönliche Weiterentwicklung miteinander verbindet. Abbildung 19 zeigt wie Aufgaben übernommen und visualisiert werden: Links eine Darstellung aus Hoda et al. (vgl. [24], S. 113), rechts ein Beispiel aus den untersuchten Fallstudien.

Abbildung 19: Vergleich Taking Task Ownership

6.2.6 Übereinstimmendes Practice 6: Group Programming

Diese Praktik gehört zur Kategorie „Encouraging Cross-Functionality“ (siehe Kapitel 3.2.5) von Hoda et al. Group Programming fördert die Zusammenarbeit im Team, indem Entwickler gemeinsam an Aufgaben arbeiten. Diese Praxis ermöglicht nicht nur den Wissensaustausch, sondern erleichtert auch die Einarbeitung neuer Teammitglieder. Teams nutzen dabei offene Arbeitsumgebungen, die eine direkte Kommunikation und gemeinsame Problemlösung fördern. Laut Hoda et al. schaffen offene Arbeitsbereiche ohne Trennwände oder separate Büros eine Umgebung, in der Tester und Entwickler effektiv zusammenarbeiten und gegenseitig von ihren Perspektiven profitieren können. Diese direkte Kommunikation spart Zeit und stärkt den Teamzusammenhalt.

In den Fallbeispielen zeigt sich diese Praxis durch verschiedene Formen der engen Zusammenarbeit: Bei komplexen Problemstellungen oder Unsicherheiten setzen sich Entwickler gezielt zusammen, um Aufgaben gemeinsam zu bearbeiten (Pair-Programming). Gleichzeitig wird durch das etablierte Vier-Augen-Prinzip sichergestellt, dass kein Code ohne Review durch eine zweite

Person in das System übernommen wird. Diese gemeinsame Arbeit unterstützt sowohl die Qualitätssicherung als auch den kontinuierlichen Wissensaustausch im Team.

Übereinstimmung mit: „Pair-Programming, 4-Augen-Prinzip und Reviews“ (siehe Punkt 6.1.3.2).

Begründung: Beide Ansätze fördern den Wissensaustausch und die Zusammenarbeit. In den Fallbeispielen wird Pair-Programming häufig genutzt, um neue Teammitglieder einzuarbeiten, Wissen zwischen Teammitglieder auszutauschen oder komplexe Probleme zu lösen. Auch Hoda et al. beschreibt, dass Group Programming den Wissensaustausch und die Einarbeitung neuer Mitglieder fördert.

6.2.7 Übereinstimmendes Practice 7: Learning Spike

Diese Praktik gehört zur Kategorie „Self-Improvement“ (siehe Kapitel 3.2.7) von Hoda et al. Ein Learning Spike bietet Teams gezielt Zeit, um neue Technologien oder Methoden zu erlernen. Diese Praxis wird besonders dann angewandt, wenn das Team vor neuen Herausforderungen steht, die spezifisches Wissen erfordern. Während der Learning Spike teilweise die Iterationsgeschwindigkeit reduziert, ermöglicht er den Teammitgliedern, sich in neuen Technologien weiterzuentwickeln und langfristig effizienter zu arbeiten.

In den Fallbeispielen wird ebenfalls betont, wie wichtig es ist, Zeit für die Einführung neuer Technologien und Verbesserungen bereitzustellen. Häufig entstehen diese Initiativen jedoch nicht durch formale Planung, sondern werden eigenverantwortlich von Teammitgliedern angestoßen – etwa um neue Tools zu evaluieren oder automatisierte Abläufe zu verbessern.

Übereinstimmung mit: „Weiterbildung durch Eigeninitiative und Teamförderung“ (siehe Punkt 6.1.5.1).

Begründung: Learning Spikes bzw. die Zeit für Verbesserungen und Technologiewechsel helfen autonomen Teams, Wissenslücken zu schließen und langfristig effizienter zu arbeiten. Die Möglichkeit, gezielt Zeit für neue Technologien und Weiterentwicklungen (zum Beispiel bei automatisierten Abläufen im Bereich Testen oder DevOps) bereitzustellen, macht diese Praktik sowohl in kleinen als auch in großen Projekten skalierbar.

6.2.8 Übereinstimmendes Practice 8: Pair-in-Need

Diese Praktik gehört zur Kategorie „Self-Improvement“ (siehe Kapitel 3.2.7) von Hoda et al. Pair-in-Need beschreibt die Zusammenarbeit zweier Teammitglieder bei der Lösung komplexer Aufgaben. Diese Praxis wird insbesondere bei herausfordernden oder designintensiven Themen eingesetzt. Laut Hoda et al. wird Pair-in-Need auf Basis von Bedarf angewandt, beispielsweise bei unvorhersehbaren oder komplexen Aufgaben. Diese Zusammenarbeit fördert den Wissensaustausch und ermöglicht es weniger erfahrenen Teammitgliedern, durch Mentoring von erfahreneren Kollegen zu lernen. Zusätzlich trägt diese Praxis dazu bei, Herausforderungen gemeinsam zu meistern und gleichzeitig den Iterationszielen näherzukommen.

In den Fallbeispielen zeigt sich ein ähnlicher Ansatz. Hier wird das 4-Augen-Prinzip genutzt, um sicherzustellen, dass komplexe Themen sorgfältig überprüft und bearbeitet werden. Pair-Programming dient als Werkzeug, um Wissen zu teilen und die Qualität der Ergebnisse zu sichern.

Übereinstimmung mit: „Pair-Programming, 4-Augen-Prinzip und Reviews“ (siehe Punkt 6.1.3.2).

Begründung: Sowohl bei Hoda et al. als auch in den Fallbeispielen zeigt sich, dass bei besonders schwierigen oder unklaren Aufgaben gezielte Zusammenarbeit im Team besonders hilfreich ist. Die Praktik „Pair-in-Need“ beschreibt eine situationsabhängige Unterstützung – zum Beispiel, wenn ein Entwickler bei einer Aufgabe nicht weiterkommt und sich gezielt mit einem Kollegen oder einer Kollegin abspricht, um gemeinsam eine Lösung zu erarbeiten. Dadurch kann Wissen direkt weitergegeben, Probleme schneller gelöst und die Qualität verbessert werden. In den Fallbeispielen zeigen sich ähnliche Muster, etwa durch Pair-Programming oder das 4-Augen-Prinzip. Diese Art der Zusammenarbeit ist flexibel einsetzbar und funktioniert sowohl in kleinen als auch in großen Projekten.

6.2.9 Übereinstimmendes Practice 9: Retrospectives

Diese Praktik gehört zur Kategorie „Self-Evaluation“ (siehe Kapitel 3.2.6) von Hoda et al. Retrospektiven finden am Ende jeder Iteration statt und dienen dazu, gemeinsam zu reflektieren, was gut lief, was verbessert werden kann, und konkrete Maßnahmen für zukünftige Iterationen abzuleiten. Hoda beschreibt, dass Retrospektiven ein essenzielles Werkzeug für die Selbstbewertung und die kontinuierliche Verbesserung der Teamarbeit sind. Diese Praxis fördert eine offene Kommunikation über Stärken und Schwächen des Teams und ermöglicht eine gezielte Weiterentwicklung.

In den Fallbeispielen werden Retrospektiven unterschiedlich angewandt. Im Fallbeispiel 1 finden Retrospektiven zwar regelmäßig statt, es fehlt jedoch manchmal an der konsequenten Umsetzung der Maßnahmen. Im Fallbeispiel 2 und 3 hingegen werden Retrospektiven als effektives Instrument genutzt, um gezielte Maßnahmen für den nächsten Sprint abzuleiten.

Übereinstimmung mit: „Regelmäßige Reflexion und Feedbackschleifen“ (siehe Punkt 6.1.3.1).

Begründung: Sowohl Hoda et al. als auch die Fallbeispiele betonen die Bedeutung von Retrospektiven für die kontinuierliche Verbesserung und die Anpassungsfähigkeit von Teams. Retrospektiven ermöglichen es, Probleme zu identifizieren und Verbesserungen einzuplanen. Die Praxis ist skalierbar und wird sowohl in kleinen autonomen Teams als auch in großen Projekten erfolgreich angewandt. Durch die regelmäßige Reflexion können Teams langfristig ihre Arbeitsweise optimieren und ihre Ziele effizienter erreichen.

6.2.10 Nicht skalierbare Best Practices

Die folgenden Praktiken von Hoda et al. finden keine Zuordnungen in den Best Practices der Fallbeispiele. Die folgenden Unterpunkte listen diese Praktiken mit einer Begründung, warum diese nicht skalierbar sind:

6.2.10.1 Multiple Perspectives

Diese Praktik aus der Kategorie „Need for Specialization“ (siehe Kapitel 3.2.4), hebt die Zusammenarbeit verschiedener Rollen innerhalb eines Teams hervor, um Perspektiven zu vereinen und bessere Lösungen zu entwickeln. Hodas Forschung zeigt, dass Selbstorganisation in Agile-Teams dazu beiträgt, formale Rollengrenzen aufzuweichen. Entwickler helfen bei Bedarf Testern, Tester lernen die Perspektiven von Entwicklern kennen, und alle Teammitglieder profitieren von einem interdisziplinären Austausch. Die Zusammenarbeit führt zu einem Lernumfeld, welches sowohl die Wissensverteilung als auch die Resilienz des Teams fördert.

In den Fallbeispielen steht jedoch die klare Rollenzuordnung im Vordergrund, wobei die Zusammenarbeit zwischen den Rollen hauptsächlich in spezifischen Kontexten wie Reviews oder Abstimmungen stattfindet. In den Fallbeispielen haben einige Teammitglieder jedoch mehrfach Rollen inne, die spezifisch auf die jeweiligen Projektanforderungen zugeschnitten sind. Diese Mehrfachrollen unterscheiden sich jedoch von der Praxis, wie sie bei Hoda beschrieben wird.

Grund für die fehlende Übereinstimmung: Während Hoda eine bewusste Aufweichung strikter Rollentrennungen beschreibt, um durch aktive Zusammenarbeit von Entwicklern, Testern und anderen Spezialisten bessere Lösungen zu erzielen, ergibt sich in den Fallbeispielen ein anderes Bild. Dort sind die Rollen in der Regel klar abgegrenzt und obwohl es natürlich Abstimmungen zwischen ihnen gibt, wurde keine systematische, gemeinsame Bearbeitung von Aufgaben über Rollengrenzen hinweg beobachtet oder erwähnt. In manchen Fällen übernehmen Personen zwar mehrere Rollen – etwa als Entwickler und Architekt – doch diese Mehrfachrollen ersetzen nicht den gezielten Perspektivenaustausch, wie er im Practice „Multiple Perspectives“ vorgesehen ist.

6.2.10.2 Self-Committing to Team Goals

In der Kategorie „Collective Decision-Making“ (siehe Kapitel 3.2.1) aus Hodas Forschung wird beschrieben, wie Teams sich selbst zu Iterationszielen verpflichten und diese durch gemeinsame Absprachen festlegen. Kunden liefern Projektanforderungen in Form von User Stories, die das Team analysiert, in Aufgaben unterteilt und basierend auf seiner Kapazität plant. Die Teams genießen dabei die Freiheit, den Entwicklungsumfang einer Iteration selbst zu definieren, und fühlen sich gleichzeitig verantwortlich, das festgelegte Ziel durch gemeinschaftliches Engagement zu erreichen.

In den Fallbeispielen hingegen erfolgt die Aufgabenplanung überwiegend durch zentrale Rollen oder Abstimmungen mit Stakeholdern, ohne dass eine teamweite Selbstverpflichtung deutlich erkennbar ist.

Grund für die fehlende Übereinstimmung: In den Fallbeispielen beteiligen sich die Teams aktiv an der Aufwandsschätzung und an der iterativen Planung einzelner Aufgaben. Die inhaltlichen Ziele einer Iteration – also was konkret umgesetzt werden soll – werden jedoch überwiegend durch zentrale Rollen oder in Abstimmung mit Stakeholdern vorgegeben. Damit fehlt die in Hodas Best Practice beschriebene zentrale Selbstverpflichtung auf ein selbst gesetztes Ziel. Zudem wirken in Softwaregroßprojekten zusätzliche Einschränkungen auf die Zielautonomie: Schnittstellen zu anderen Teams, gemeinsame Releasezyklen und koordinationsbedingte Abhängigkeiten reduzieren die Freiheit, Iterationsziele vollständig unabhängig zu definieren. Diese strukturellen Rahmenbedingungen erschweren ein kollektives Commitment, wie es in selbstorganisierten Kleinteams möglich ist, und verhindern damit eine vollständige Entsprechung zum Practice „Self-Committing to Team Goals“ im Sinne von Hoda.

6.2.10.3 Collectively Deciding Team Norms & Principles

Diese Praktik aus der Kategorie „Collective Decision-Making“ (siehe Punkt 3.2.1.2 in Kapitel 3.2.1), wird die gemeinsame Festlegung von Arbeitsnormen von Hoda beschrieben. Die Teams einigen sich auf organisatorische Prinzipien wie Arbeitszeiten, Fehlertoleranzen und Entwicklungsrichtlinien, die die Zusammenarbeit und Zielerreichung fördern. Dieser Prozess wird als integraler Bestandteil der Teamkultur angesehen.

In den Fallbeispielen zeigt sich ein anderer Ansatz. Hier werden technische Standards und Richtlinien teils durch das Team ergänzt, beispielsweise durch Checklisten oder Review-Prozesse. Jedoch fehlt eine umfassende teamweite Festlegung auf organisatorische Normen, wie sie bei Hoda beschrieben wird.

Grund für die fehlende Übereinstimmung: In den Fallbeispielen werden technische Standards und Richtlinien teils durch das Team ergänzt, z. B. durch Checklisten oder Review-Prozesse. Jedoch fehlt eine teamweite Festlegung auf organisatorische Normen, wie Arbeitszeiten oder Defekttoleranz, die Hoda beschreibt. Der Fokus liegt stärker auf technischen Vorgaben durch Architekten oder Teamleiter.

6.2.10.4 Rotation

Die Best Practice Rotation aus der Kategorie „Encouraging Cross-Functionality“ (siehe Kapitel 3.2.5) beschreibt den regelmäßigen Wechsel von Verantwortlichkeiten über Teamgrenzen hinweg, um die Wissensbasis zu erweitern und neue Fähigkeiten zu entwickeln. In der Forschung von Hoda et al. wird Rotation als strategisches Mittel beschrieben, um Wissen durch direkte Kommunikation zu teilen.

In den Fallbeispielen gibt es gewisse Ähnlichkeiten, wie z. B. die Möglichkeit von Themen- oder Rollenwechseln (siehe Kapitel 6.1.5.2 und 6.1.5.3). Diese basieren jedoch auf individueller Initiative und werden nicht als routinemäßiges Practice innerhalb der Teams umgesetzt.

Grund für die fehlende Übereinstimmung: Das von Hoda et al. beschriebene Konzept der Rotation umfasst einen gezielten, strukturierten und regelmäßig praktizierten Wechsel von Aufgaben und Verantwortlichkeiten über Teamgrenzen hinweg. In den Fallbeispielen hingegen liegt der Schwerpunkt auf der teaminternen Zusammenarbeit. Rollen- oder Themenwechsel erfolgen dort lediglich bei Bedarf – etwa, wenn personelle Engpässe entstehen oder individuelle Interessen be-

stehen. Solche Wechsel werden situativ angestoßen, jedoch nicht als kontinuierliche oder strategisch geplante Maßnahme zur Wissensverbreitung etabliert. Eine Rotation im Sinne von Hoda konnte daher nicht beobachtet werden.

Kapitel-nummer (Hoda et al.)	Best Practice (Hoda et al.)	Matched Best Practice	Kapitel-nummer	Kommentar
3.2.1.1	Collective Estimation & Planning	Iterative Verfeinerung der Schätzungen durch das gesamte Team	6.1.13.1	
3.2.1.2	Collectively Deciding Team Norms & Principles			Kein Match – siehe Kapitel 6.2.10.3
3.2.1.3	Self-Committing to Team Goals			Kein Match – siehe Kapitel 6.2.10.2
3.2.2.1	Using Story Board	Toolunterstützter Workflow	6.1.1.1	
3.2.2.2	Taking Task Ownership	Förderung von Eigenverantwortung durch selbstbestimmte Aufgabenwahl	6.1.9.2	
3.2.3.1	Daily Standup Meetings	Regelmäßige synchrone Abstimmung im Team	6.1.1.2	
3.2.3.2	Information Radiators	Toolunterstützter Workflow	6.1.1.1	
3.2.4.1	Multiple Perspectives			Kein Match – siehe Kapitel 6.2.10.1
3.2.5.1	Group Programming	Pair-Programming, 4-Augen-Prinzip und Reviews	6.1.3.2	
3.2.5.2	Rotation			Kein Match – siehe Kapitel 6.2.10.4
3.2.6.1	Retrospectives	Regelmäßige Reflexion und Feedbackschleifen	6.1.3.1	
3.2.7.1	Pair-in-Need	Pair-Programming, 4-Augen-Prinzip und Reviews	6.1.3.2	
3.2.7.2	Learning Spike	Weiterbildung durch Eigeninitiative und Teamförderung	6.1.5.1	

Tabelle 25: Übersicht skalierbarer Best Practices

6.3 Zusammenfassung der Themen als Kategorien

Die übergreifende thematische Analyse der drei Fallstudien identifiziert als Ergebnis 13 Hauptthemen (siehe Kapitel 5.2.1), die aus Tabelle 13 abgeleitet sind. Die Tabelle ist das Ergebnis einer Überschneidung aller Themen und Codes, welche in allen drei Fallbeispielen gemeinsam auftreten. Die Themen werden in den folgenden Punkten beschrieben und als Kategorien übernommen. Eine Kategorie ist eine Sammlung mehrerer Praktiken, deren Kriterien für eine Zuordnung im

Punkt 6.4 beschrieben sind. Die in diesem Kapitel beschriebenen Kategorien fassen mehrere Praktiken zusammen, die gemeinsam darauf abzielen, die jeweiligen thematischen Schwerpunkte effektiv zu unterstützen und zu verbessern. Jede einzelne Praxis innerhalb dieser Kategorien trägt aktiv zur Erreichung der dargestellten Ziele bei und ermöglicht eine gezielte Optimierung der Teamarbeit in Softwaregroßprojekten. Dieses Kapitel liefert die Antwort auf die Forschungsfrage RQ2: „In welche Kategorien können die identifizierten Praktiken unterteilt werden?“.

6.3.1 Transparenz

Transparenz ist eine grundlegende Voraussetzung für eine effektive Zusammenarbeit in autonomen Teams. Durch Praktiken wie Kanban-Boards, Ticket-Systeme (z. B. Jira) und tägliche Abstimmungen (Dailies) werden Informationen zu Aufgaben, Fortschritt und auftretenden Problemen kontinuierlich sichtbar gemacht. Dadurch erhalten alle Teammitglieder transparente Einsicht in den aktuellen Stand der Arbeit, was Abstimmungen erleichtert, Entscheidungsprozesse beschleunigt und potenzielle Engpässe frühzeitig sichtbar macht.

6.3.2 Kommunikation

Eine klare und effektive Kommunikation ist notwendig für den Erfolg autonomer Teams. Sie umfasst formelle als auch informelle Interaktionsformen, die sicherstellen, dass alle Teammitglieder auf dem gleichen Informationsstand sind. Dazu zählen direkte Gespräche, Meetings, Telefonate, Chat-Tools und Dokumentationsplattformen. Während Meetings strukturierte Diskussionen ermöglichen, fokussiert sich diese Kategorie auf spontane Abstimmungen, schnelle Rückfragen und informelle Austauschformate, die den reibungslosen Ablauf von Arbeitsprozessen unterstützen.

Abgrenzung zur Kategorie Meetings (6.3.7):
Meetings sind eine organisierte und geplante Form der Kommunikation, die regelmäßig zu fixen Zeiten stattfindet oder bedarfsabhängig geplant wird. Die allgemeine Kommunikation hingegen umfasst dynamische und oft ad-hoc stattfindende Abstimmungen, die für den laufenden Arbeitsprozess essenziell sind. Während Meetings eher der strategischen oder tiefergehenden Diskussion dienen, zielt Kommunikation auf den kontinuierlichen und informellen Austausch im Tagesgeschäft ab.

6.3.3 Reflexion & Lernprozess

Regelmäßige Reflexionen, wie Retrospektiven oder Reviews, sind wichtige geplante Einheiten für autonome Teams, um sich zu verbessern. Sie ermöglichen es positive als auch negative Erfahrungen zu analysieren und den zukünftigen Workflow zu optimieren.

6.3.4 Wissensverteilung und Support

Autonome Teams profitieren stark von einer gleichmäßigen Verteilung des Wissens. Maßnahmen wie Pair-Programming, Code Reviews und Dokumentation von Entscheidungen und Umsetzungen fördern die Vermeidung von personenbezogenem Wissen („Single Point of Failure“). Diese Praktiken helfen auch, Risiken wie zum Beispiel den Ausfall einzelner Mitglieder zu minimieren.

6.3.5 Fortbildung

Die Möglichkeit zur Fortbildung und Entwicklung ist wichtig für Mitglieder autonomer Teams, damit sich diese ihren Interessen widmen können. Zeitfenster für technologische Experimente, der Erwerb von Zertifikaten und gezielte Weiterbildungsmaßnahmen fördern nicht nur die individuellen Kompetenzen, sondern stärken auch die Innovationsfähigkeit des gesamten Teams.

6.3.6 Richtlinien, Grenzen, Normen und Prinzipien

Autonome Teams folgen klar definierten Standards, die oft durch technische Architekten oder Teamleiter vorgegeben werden. Diese Richtlinien gewährleisten Einheitlichkeit, Codequalität und eine bessere Zusammenarbeit. In Softwaregroßprojekten werden diese Richtlinien meist über alle Teams hinweg, projektweit von einem technischen Architekten vorgegeben. Tools wie zum Beispiel Checkstyle oder SonarQube unterstützen die Einhaltung dieser Standards und machen Vorgaben für alle transparent und nachvollziehbar. Trotzdem ist es wichtig, dass das autonome Team selbst eine Teamkultur bilden und Standards für sich definieren und verfeinern kann.

6.3.7 Meetings

Meetings wie Dailies, Weeklies oder spezifische Teamleiter-Meetings sind unerlässlich für den Informationsaustausch und die Koordination – sowohl innerhalb autonomer Teams als auch teamübergreifend. Sie fördern Transparenz, ermöglichen eine kontinuierliche Abstimmung und helfen dabei, Herausforderungen frühzeitig zu identifizieren und zu lösen, bevor sie sich zu größeren Problemen ausweiten.

Meetings sind im Gegensatz zur alltäglichen Kommunikation gezielt geplante und strukturierte Treffen. Sie helfen den autonomen Teams, sich regelmäßig über den Fortschritt, sowie mögliche Probleme und wichtige Entscheidungen abzustimmen.

6.3.8 Fortschrittsmessung

Autonome Teams setzen gezielt messbare Indikatoren und Tools zur Aufgabenverwaltung ein, um sowohl den Status einzelner Tasks als auch den Gesamtfortschritt des Projekts kontinuierlich sichtbar zu machen. Regelmäßige Auswertungen dieser Indikatoren ermöglichen es den Teams, realistische Prognosen zu erstellen, zeitliche Abweichungen frühzeitig zu erkennen und notwendige Anpassungen schnell umzusetzen. Durch diese kontinuierliche Fortschrittsüberwachung können Risiken reduziert, Ressourcen optimal genutzt und Projektergebnisse zuverlässig erreicht werden.

6.3.9 Auswahl und Zuteilung von Tasks

Die Aufgabenverteilung in autonomen Teams erfolgt entweder durch zentrale Koordination (z. B. durch den Teamleiter) oder durch Selbstzuweisung der Mitglieder. Dabei wird auf Transparenz und klare Nachverfolgbarkeit geachtet. Die Zuteilung orientiert sich häufig an den Kompetenzen, der zeitlichen Verfügbarkeit und den Interessen der Teammitglieder. Gleichzeitig bleibt die Autonomie jedes Einzelnen gewahrt, sich Aufgaben eigenverantwortlich zu nehmen oder im gegenseitigen Einvernehmen umzuschichten.

6.3.10 Taskänderungen

Dynamische Änderungen, sowohl technischer als auch fachlicher Natur, werden in autonomen Teams validiert, priorisiert und transparent kommuniziert. Während fachliche Änderungen häufig Abstimmungen mit Stakeholdern erfordern, können kleinere technische Anpassungen, wie kleinere Bugfixes oder Optimierungen, flexibel und ohne großen Aufwand von den autonomen Teams sofort umgesetzt werden.

6.3.11 Team Events & Belohnungen

Allgemeine Teamevents wie gemeinsames Essen oder Outdoor-Aktivitäten stärken den sozialen Zusammenhalt und fördern das gegenseitige Vertrauen innerhalb autonomer Teams. Solche Veranstaltungen bieten eine Möglichkeit, Erfolge zu feiern und neue Mitglieder besser zu integrieren.

6.3.12 Komplexe Themen

Bei der Bearbeitung komplexer Themen ziehen autonome Teams häufig Expertenmeinungen heran oder nutzen Praktiken wie Pair-Programming. Als Erstes werden Experten innerhalb des Teams herangezogen. Sollte das Know-how nicht vorhanden sein, wird auf externe Experten zurückgegriffen. Dies stellt sicher, dass auch schwierige Herausforderungen effizient bewältigt werden können. Regelmäßige Abstimmungen fördern dabei den Wissensaustausch und die Qualität der Ergebnisse.

6.3.13 Aufwandsschätzung

Aufwands- und Komplexitätsschätzungen erfolgen in autonomen Teams häufig kollaborativ. Praktiken wie Planning Poker mit Story Points helfen, die Ressourcenplanung zu optimieren und realistische Zeitpläne zu erstellen. Die Einbindung aller Mitglieder bei Schätzungen erhöht die Genauigkeit und Akzeptanz der Ergebnisse.

6.4 Kriterien für die 13 Kategorien

Dieser Abschnitt beantwortet die Forschungsfrage RQ3: „Auf Basis welcher Kriterien kann eine Kategorisierung stattfinden?“ (vgl. Punkt 1.3) und ergänzt die bisherigen Ausführungen der vorherigen Kapitel 6.1 bis 6.3, indem die Kriterien für die Zuordnung der identifizierten Best Practices zu den 13 Hauptkategorien beschrieben werden.

Kriterien für eine Zuordnung von Best Practices zu einer Kategorie umfassen verschiedene Aspekte:

- **Inhaltliche Relevanz:**
  Der Inhalt des Best Practices trägt zu der Zielsetzung und Wirkung dieser Kategorie bei.
- **Ableitung aus Codes der thematischen Analyse:**
  Die Zuordnung basiert auf spezifischen Codes der thematischen Analyse nach Braun & Clarke der drei Fallbeispiele und dem Kontext der Interviews. Die Häufigkeit der Codes für Praktiken und das Vorkommen über alle drei Fallbeispiele stützen eine Zuordnung. Dabei wurde geprüft, wie oft bestimmte Begriffe in den kodierten Segmenten der Interviews auftraten und in welchen Zusammenhängen sie erwähnt wurden. Diese Methode stellt sicher, dass die Praktiken nicht nur theoretisch, sondern auch empirisch fundiert den Kategorien zugeordnet werden.
- **Konsistenz innerhalb der Kategorie:**

Best Practices dieser Kategorie ergänzen sich gegenseitig und ergeben ein konsistentes Bild und stehen nicht im Widerspruch zueinander.

6.4.1 Kriterien der Kategorie „Transparenz“

Die Transparenz (siehe Punkt 6.3.1) beschreibt die Nachvollziehbarkeit und Sichtbarkeit von Informationen, Prozessen und Fortschritten innerhalb eines autonomen Teams. Praktiken in dieser Kategorie tragen inhaltlich dazu bei, dass ein transparenter Zugang zu Informationen gewährleistet wird. Sie basieren auf den Codes der thematischen Analyse (z. B. „Software & Ticketverwaltung“, „Dailies (Transparenz)“) und fördern Offenheit in Kommunikation und Dokumentation. Praktiken innerhalb der Kategorie ergänzen sich, indem sie unterschiedliche Aspekte der Transparenz abdecken, wie die Nutzung von Tools, regelmäßige Besprechungen und die Einbindung mehrerer Personen pro Thema. Kriterien, die erfüllt sein müssen:

- **Offene Kommunikation:** Eine offene Kommunikation der Teammitglieder muss möglich sein, um Fortschritte und Herausforderungen zu teilen.
- **Toolunterstützte Nachvollziehbarkeit von Aufgaben und Fortschritten:** Tools zur Nachvollziehbarkeit von Aufgaben und Prozessen müssen eingesetzt werden. (z.B. „Toolunterstützter Workflow“, siehe Punkt 6.1.1.1)
- **Nachvollziehbare und standardisierte Dokumentation:** Entscheidungen und deren Gründe müssen für alle zugänglich dokumentiert sein. (z.B. „Standardisierte Protokollierung und Dokumentation“, siehe Punkt 6.1.1.4)
- **Regelmäßige Abstimmungen:** Regelmäßige Besprechungen mit Teammitgliedern müssen Transparenz über Fortschritte und Hindernisse schaffen. (z.B. „Regelmäßige synchrone Abstimmung im Team“, siehe Punkt 6.1.1.2)
- **Einbindung mehrerer Personen:** Mehrere Personen sollen über ein Thema Bescheid wissen.

6.4.2 Kriterien der Kategorie „Kommunikation“

Kommunikation umfasst den zielgerichteten Austausch von Informationen, um eine reibungslose Zusammenarbeit zu gewährleisten. Dies kann durch persönliche Gespräche, digitale Tools oder dokumentierten Austausch erreicht werden. Kriterien für die Zuordnung sind wie folgt:

- **Direkte Kommunikation fördern:** Einfache und direkte Kommunikationswege, wie durch physische Nähe in Büros oder digitale Tools, sind essenziell für schnelle Abstimmungen. (z.B. „Effiziente Abstimmung durch direkte Kommunikation“, siehe Punkt 6.1.2.1)

- **Verfügbarkeit von Kommunikationsmittel:** Telefon, Chats und Videokonferenzen ermöglichen effektiven Austausch, insbesondere in verteilten Teams.
- **Zentrale Dokumentation:** Tool wie Wikis gewährleisten, dass Informationen langfristig zugänglich bleiben und Entscheidungen nachvollzogen werden können. (z.B. „Toolunterstützter Workflow“, siehe Punkt 6.1.2.2)

6.4.3 Kriterien der Kategorie „Reflexion & Lernprozess“

Reflexion beschreibt die Fähigkeit eines Teams, regelmäßig die eigene Arbeit zu analysieren und Verbesserungen abzuleiten.

- **Geplante Reflexion:** Praktiken, die regelmäßigen Retrospektiven oder Reviews als Bestandteil des Workflows integrieren. (z.B. „Regelmäßige Reflexion und Feedbackschleifen“, siehe Punkt 6.1.3.1)
- **Lernorientierung:** Praktiken, die darauf abzielen, aus Erfahrungen zu lernen und den Workflow zu optimieren.

6.4.4 Kriterien der Kategorie „Wissensverteilung & Support“

Wissensverteilung umfasst die Vermeidung von Wissensinseln und die Förderung des Wissensaustauschs innerhalb des Teams.

- **Vermeidung von Wissensinseln und Förderung von Wissensaustausch:** Praktiken, die sicherstellen, dass Wissen innerhalb des Teams geteilt wird. (z.B. Mehrere Themenverantwortliche zur Wissensverteilung“, siehe Punkt 6.1.4.4 / „Pair-Programming, 4-Augen-Prinzip und Reviews“, siehe Punkt 6.1.4.1)
- **Unterstützung bei Engpässen:** Praktiken, die sicherstellen, dass bei Abwesenheit von Teammitgliedern keine Wissenslücken entstehen.

6.4.5 Kriterien der Kategorie „Fortbildung“

Fortbildung umfasst Maßnahmen zur kontinuierlichen Weiterentwicklung von Teammitgliedern, um individuelle Kompetenzen zu stärken und die Innovationsfähigkeit des Teams zu fördern. Die Kriterien für die Zuordnung sind:

- **Förderung individueller Entwicklung:** Praktiken müssen gezielt darauf abzielen, die Fähigkeiten und Kompetenzen der Teammitglieder zu erweitern, z. B. durch Weiterbildung, Schulungen oder Zertifikate. (z.B. „Weiterbildung durch Eigeninitiative und Teamförderung“, siehe Punkt 6.1.5.1)
- **Anpassung an technologische Veränderungen:** Praktiken sollten Zeitfenster für Experimente oder den Umgang mit neuen Technologien und Versionen ermöglichen. Sie sollen Teammitgliedern ermöglichen zwischen Themen und Zuständigkeiten innerhalb oder außerhalb des Teams zu rotieren. (z.B. „Themenwechsel zur individuellen Weiterentwicklung“, siehe Punkt 6.1.5.2 / „Rollenwechsel zur Entwicklung individueller Fähigkeiten“, siehe Punkt 6.1.5.3)

6.4.6 Kriterien der Kategorie „Richtlinien, Grenzen, Normen und Prinzipien“

Diese Kategorie umfasst klar definierte Standards und Vorgaben, die eine einheitliche Arbeitsweise im Team und projektübergreifend sicherstellen. Sie fördern Konsistenz, Qualität und Zusammenarbeit. Kriterien für die Zuordnung sind:

- **Definition technischer und organisatorischer Qualitätsrichtlinien:** Praktiken müssen zur Definition und Einhaltung von technischen, organisatorischen oder projektübergreifenden Richtlinien beitragen, z. B. durch Coding-Standards oder Architekturvorgaben. (z.B. „Dokumentation und Festlegung von Richtlinien und Prozessen“, siehe Punkt 6.1.6.1)
- **Transparenz der Vorgaben:** Standards und Normen müssen dokumentiert und für alle Teammitglieder zugänglich sein, z. B. in Confluence oder ähnlichen Plattformen.
- **Erweiterung teamindividueller Qualitätsrichtlinien:** Praktiken müssen sicherstellen, dass Teams ihre spezifischen Normen ergänzen und anpassen können, ohne den Rahmen der projektweiten Richtlinien zu verlassen. (z.B. „Teamspezifische Erweiterungen von Qualitätsstandards“, siehe Punkt 6.1.6.2)

6.4.7 Kriterien der Kategorie „Meetings“

Meetings sind essentielle Instrumente zur Abstimmung, Koordination und Entscheidungsfindung innerhalb und zwischen Teams. Sie dienen der Sicherstellung von Transparenz, der Lösung von Herausforderungen und der Planung von Aufgaben. Die Kriterien für die Zuordnung sind:

- **Strukturierte, regelmäßige Durchführung:** Meetings müssen klar strukturiert sein, um effektiv zu sein, z. B. durch festgelegte Agenden, Zeitrahmen und Protokollierung. (z.B. „Regelmäßige synchrone Abstimmung im Team“, siehe Punkt 6.1.7.1 / „Wöchentliche Abstimmung zwischen Teams“, siehe Punkt 6.1.7.2)
- **Zielgerichtete Kommunikation:** Meetings müssen dazu beitragen, Herausforderungen zu identifizieren und Lösungen gemeinsam zu entwickeln, ohne dabei die Effizienz zu beeinträchtigen.

6.4.8 Kriterien der Kategorie „Fortschrittsmessung“

Fortschrittsmessung umfasst die Erfassung und Darstellung des Fortschritts auf Aufgaben- und Projektebene, um Transparenz und Planbarkeit zu gewährleisten. Die Kriterien für die Zuordnung sind:

- **Messbarkeit der Fortschritte:** Praktiken müssen klare Indikatoren definieren, um Fortschritte auf Aufgaben- und Projektebene nachvollziehbar zu machen, z. B. durch Ticketstatus oder Prozentangaben.
- **Dokumentation von Fortschritten:** Fortschritte müssen erfasst und für alle Teammitglieder sichtbar dokumentiert werden, z. B. in Tools wie Jira oder auf Kanban-Boards. (z.B. „Toolunterstützter Workflow“, siehe Punkt 6.1.8.2)
- **Regelmäßige Evaluierung:** Praktiken müssen Fortschritte regelmäßig überprüfen, z. B. durch Meetings oder Dashboards, um frühzeitig Risiken oder Verzögerungen zu erkennen. (z.B. „Regelmäßige Fortschrittsbesprechungen“, siehe Punkt 6.1.8.1 / „Toolunterstützter Workflow“, siehe Punkt 6.1.8.2)

6.4.9 Kriterien der Kategorie „Auswahl und Zuteilung von Tasks“

Diese Kategorie beschreibt, wie Aufgaben in autonomen Teams verteilt werden, sei es durch zentrale Koordination oder durch Selbstzuweisung der Teammitglieder. Ziel ist es, Transparenz und Effizienz bei der Verteilung der Aufgaben zu gewährleisten. Die Kriterien für die Zuordnung:

- **Transparenz der Zuteilung:** Praktiken müssen sicherstellen, dass die Vergabe der Aufgaben für alle Teammitglieder nachvollziehbar ist, z. B. durch die Verwendung von Tools wie Jira.
- **Berücksichtigung individueller Kompetenzen:** Praktiken müssen die Fähigkeiten, Interessen und Kapazitäten der Teammitglieder berücksichtigen, um Aufgaben optimal zuzuteilen.
- **Flexibilität in der Zuteilung:** Praktiken sollen sowohl eine zentrale Zuteilung als auch eine individuelle Zuteilung der Aufgaben ermöglichen. (z.B. „Koordinierten Aufgabenverteilung

für kritische und spezialisierte Aufgaben“, siehe Punkt 6.1.9.1 / „Förderung von Eigenverantwortung durch selbstbestimmte Aufgabenwahl“, siehe Punkt 6.1.9.2)

6.4.10 Kriterien der Kategorie „Task-Änderungen“

Diese Kategorie beschreibt den Umgang mit dynamischen Änderungen von Aufgaben, sei es durch fachliche oder technische Anforderungen. Ziel ist es, flexibel und effizient auf Änderungen zu reagieren, ohne den Projektfluss zu beeinträchtigen. Die Kriterien für die Zuordnung sind:

- **Strukturierte Validierung:** Praktiken bewerten und priorisieren Änderungen um eine Einplanung und effiziente Umsetzung zu ermöglichen. (z.B. „Strukturierte Validierung und Priorisierung von Aufgabenänderungen“, siehe Punkt 6.1.10.1)
- **Schnelle Umsetzung unkritischer Aufgabenänderungen:** Praktiken sollten sicherstellen, dass kleinere Änderungen flexibel und ohne unnötige Bürokratie umgesetzt werden können. (z.B. „Schnelle Umsetzung unkritischer Änderungen zur Vermeidung bürokratischer Hürden“, siehe Punkt 6.1.10.2)

6.4.11 Kriterien der Kategorie „Team Events & Belohnungen“

Diese Kategorie umfasst Maßnahmen, die den sozialen Zusammenhalt im Team stärken und die Motivation der Teammitglieder fördern, z. B. durch gemeinsame Aktivitäten oder die Anerkennung von Leistungen. Die Kriterien für die Zuordnung sind:

- **Förderung des Teamzusammenhalts:** Praktiken sollten darauf abzielen, das Vertrauen und die Zusammenarbeit im Team zu stärken, etwa durch informelle oder soziale Aktivitäten. (z.B. „Förderung des Teamzusammenhalts durch Teamevents“, siehe Punkt 6.1.11.1)
- **Anerkennung von Leistungen:** Praktiken müssen sicherstellen, dass die Leistungen der Teammitglieder wertgeschätzt und gewürdigt werden.
- **Integration neuer Mitglieder:** Praktiken sollten neue Teammitglieder in das Team integrieren und die Zusammenarbeit fördern.

6.4.12 Kriterien der Kategorie „Komplexe Themen“

Diese Kategorie beschreibt den Umgang mit anspruchsvollen Aufgaben, die eine vertiefte Zusammenarbeit und Expertise erfordern, um eine Lösung zu erzielen. Kriterien für die Zuordnung sind:

- **Einbindung von Expertise:** Praktiken müssen sicherstellen, dass bei komplexen Themen relevante Fachkenntnisse hinzugezogen werden, sei es durch interne oder externe Experten. (z.B. „Strukturierte Teamabstimmung und Validierung bei komplexen Aufgaben“, siehe Punkt 6.1.12.2)
- **Förderung der Zusammenarbeit:** Praktiken sollten darauf abzielen, die gemeinsame Bearbeitung von Aufgaben zu unterstützen, z. B. durch Pair-Programming oder spezialisierte Abstimmungen sowie Reviews für Kontrolle und Validierung der Umsetzung. (z.B. „Pair-Programming, 4-Augen-Prinzip und Reviews“, siehe Punkt 6.1.12.1)
- **Strukturierte Problemlösung:** Praktiken müssen klare Prozesse für die Analyse und Lösung komplexer Fragestellungen etablieren.

6.4.13 Kriterien der Kategorie „Aufwandsschätzung“

Die Aufwandsschätzung beschreibt die Prozesse zur Bewertung und Schätzung des Arbeitsaufwands für Aufgaben. Ziel ist es, realistische Zeitpläne und Ressourcenzuweisungen zu ermöglichen. Die Kriterien für die Zuordnung sind:

- **Kollaborative Schätzung:** Praktiken müssen sicherstellen, dass die Schätzung des Arbeitsaufwands im Team gemeinsam und transparent durchgeführt wird. (z.B. „Iterative Verfeinerung der Schätzungen durch das gesamte Team“, siehe Punkt 6.1.13.1)
- **Einsatz etablierter Schätzmethoden:** Praktiken sollten Techniken wie Planning Poker oder die Aufteilung in Subtasks verwenden, um den Schätzprozess zu strukturieren und einen einheitlichen Prozess zu etablieren. (z.B. „Aufteilung in Subtasks zur besseren Planung“, siehe Punkt 6.1.13.2)
- **Berücksichtigung von Erfahrungswerten:** Praktiken sollten auf vorhandenen Erfahrungswerten der Teammitglieder basieren, um realistische Vorhersagen zu ermöglichen.

6.5 Aktualisierter Abgleich der Ergebnisse im State of the Art 2024/2025

In diesem Kapitel werden die Ergebnisse dieser Arbeit nochmals aus einer übergeordneten Perspektive betrachtet, indem sie mit zentralen Erkenntnissen aktueller Forschung aus den Jahren 2024 und 2025 verglichen werden. Ziel ist es, die in den Fallbeispielen identifizierten skalierbaren Best Practices in Beziehung zu neueren theoretischen Ansätzen zur agilen Organisationsentwicklung zu setzen und deren Anschlussfähigkeit an den aktuellen Forschungsstand zu überprüfen.

Der Fachartikel „Scaling or growing agile? Proposing a manifesto for agile organization development“ von Bremer et al. (2025) [56] beschreibt ein reales Beispiel einer agilen Skalierungsinitiative bei Zenseact, bei dem der Übergang von einer stark auf Teamautonomie und informelle Strukturen ausgerichteten Arbeitsweise hin zu einem SAFe-inspirierten Rahmen zu Spannungen und Zielkonflikten führt. Die dort entwickelten Prinzipien für eine „agile Organisationsentwicklung“ bieten eine interessante Vergleichsbasis zu den in dieser Arbeit identifizierten Best Practices, da sie konkrete Herausforderungen und Lösungsansätze aus einem organisatorisch umfangreichen Projektkontext adressieren.

Die folgende Tabelle 26 stellt die fünf Prinzipien aus dem vorgeschlagenen „Manifest für agile Organisationsentwicklung“ den jeweils zugeordneten Best Practices aus Kapitel 6.1 gegenüber [56].

Prinzip (Bremer et al., 2025)	Zentrale Inhalte	Passende Best Practices aus Kapitel 6.1	Einordnung
1. Depart from the inseparability of freedom and responsibility	(Selbstorganisation braucht klare Verantwortlichkeiten – sonst bleiben Aufgaben liegen oder werden top-down vergeben. Lösung: direkte Abstimmung und Transparenz	6.1.9.2 Förderung von Eigenverantwortung durch selbstbestimmte Aufgabenwahl 6.1.1.1 Toolunterstützter Workflow 6.1.10.1Strukturierte Validierung und Priorisierung von Aufgabenänderungen 6.1.2.1 Effiziente Abstimmung durch direkte Kommunikation	Es wurden mehrere passende Praktiken gefunden
2. Nurture an environment for people to thrive and share	Aufbau einer Kultur des Vertrauens, der Offenheit und der geteilten Verantwortung	6.1.11.1 Förderung des Teamzusammenhalts durch Teamevents 6.1.2.1 Effiziente Abstimmung durch direkte Kommunikation	Teilweise passend – soziales Klima wird gefördert, jedoch ohne Vorrang vor Projektzielen.
3. Develop people through practical experience	Lernen durch praktische Zusammenarbeit statt Theorie	6.1.13.1 Iterative Verfeinerung der Schätzungen durch das gesamte Team 6.1.12.2 Strukturierte Teamabstimmung und Validierung bei komplexen Aufgaben 6.1.3.2 Pair-Programming, 4-Augen-Prinzip und Reviews	Starke Übereinstimmung, da Lernen durch Teamarbeit in allen Fallbeispielen umgesetzt wird
4. Handle complexity with simplicity	Fokus auf einfache Werkzeuge und Prinzipien	6.1.1.1 Toolunterstützter Workflow 6.1.13.2 Aufteilung in Subtasks zur besseren Planung	Passend im Sinne schlanker Prozesse und klarer Aufgabenzerlegung
5. Grow the organization at its own tempo of trust	Organisationen sollen sich im eigenen Tempo entwickeln – nicht durch beschleunigtes Skalieren; Vertrauen als Grundlage		Kein direkter Bezug – dieses Prinzip zielt auf strategisches Organisationswachstum und ist daher außerhalb des empirischen Fokus dieser Arbeit

Tabelle 26: Beziehung der Principles von Bremer et al. mit den Ergebnissen

Der Fachartikel „Investigating Communities of Practice in Large-Scale Agile Software Development: An Interview Study“ von Tobisch et al. [57] untersucht die Rolle sogenannter Communities of Practice (CoPs) in groß angelegten agilen Entwicklungsumgebungen. CoPs sind freiwillige, rollenübergreifende Zusammenschlüsse von Mitarbeitenden mit geteiltem fachlichem Interesse. Sie dienen dem Wissensaustausch, der kollaborativen Weiterentwicklung sowie der Etablierung informeller Standards und Praktiken. Die Studie basiert auf 39 Interviews aus 18 Organisationen, die Frameworks wie SAFe, LeSS oder das Spotify-Modell einsetzen.

Die Autoren zeigen, dass Communities of Practice (CoPs) auf vielfältige Weise zur agilen Transformation beitragen – insbesondere durch Stärkung der Eigenverantwortung, gezielten Wissensaustausch, teamübergreifende Abstimmung und informelle Koordination. Diese Wirkungen überschneiden sich mit mehreren der in dieser Arbeit identifizierten Best Practices aus Kapitel 6.1. Die Tabelle 27 zeigt, in welchen Bereichen sich diese Gemeinsamkeiten konkret manifestieren – etwa im Hinblick auf Empowerment, Wissensverteilung oder organisationsweites Alignment.

CoP-Ziel (nach Tobisch et al.)	Relevante Best Practices aus Kapitel 6.1	Begründung / Bezug und Abgrenzung
Fostering Empowerment	6.1.9.2 Förderung von Eigenverantwortung durch selbstbestimmte Aufgabenwahl	Beide fördern Eigenverantwortung. Die Best Practice bezieht sich auf selbstständige Aufgabenwahl im Teamkontext. CoPs schaffen übergeordnete Strukturen, in denen Mitarbeitende sich freiwillig einbringen, Verantwortung übernehmen und durch Austausch mit Gleichgesinnten fachlich weiterentwickeln können. Die Best Practice kann als Umsetzung innerhalb eines solchen CoP-Verständnisses gesehen werden.
Promoting Knowledge Sharing	6.1.4.1 Pair-Programming, 4-Augen-Prinzip und Reviews 6.1.4.4 Mehrere Themenverantwortliche zur Wissensverteilung	Beide Praktiken fördern kontinuierlichen Wissensaustausch im Alltag. CoPs bieten dafür einen strukturellen Rahmen auf übergeordneter Ebene, z. B. durch thematische Gruppen. Die genannten Praktiken setzen diesen Austausch direkt im Arbeitsprozess um.
Distributing Information	6.1.1.1 Toolunterstützter Workflow 6.1.1.4 Standardisierte Protokollierung und Dokumentation	CoPs dienen in der Studie u. a. dazu, Informationen rollen- oder themenspezifisch weiterzugeben und transparent zu machen. Die Best Practices adressieren dies durch systematische Dokumentation, Tool-Nutzung und Statusverfolgung im Alltag und fördern den transparenten Zugang zu diesen Informationen.
Fostering People Development	6.1.5.1 Weiterbildung durch Eigeninitiative und Teamförderung 6.1.3.2 Pair-Programming, 4-Augen-Prinzip und Reviews	Die CoPs ermöglichen fachliches Wachstum durch Austausch und Vernetzung. Die Best Practices setzen dies durch gezielte Weiterbildung, Zertifizierungen und kollaborative Formate um. Pair-Programming und Reviews sind Beispiele für die Umsetzung innerhalb des Teams.
Promoting Collaboration	6.1.2.1 Effiziente Abstimmung durch direkte Kommunikation	CoPs fördern Zusammenarbeit über Teams und Rollen hinweg. Die Best Practices fokussieren

	6.1.12.2 Strukturierte Teamabstimmung und Validierung bei komplexen Aufgaben	auf die Abstimmung mit Teilnehmern innerhalb und außerhalb des autonomen Teams.
Aligning Across the Organization	6.1.7.2 Wöchentliche Abstimmung zwischen Teams 6.1.7.3 Rollenspezifische Teamleiter-Meetings zur fachlichen Koordination	Beide Ansätze dienen der organisationsweiten Abstimmung. Die Best Practices setzen den Austausch über verschiedene Teams und Bereiche über regelmäßige Meetings um.
Supporting the Agile Transformation	6.1.5.1 Weiterbildung durch Eigeninitiative und Teamförderung	CoPs können laut Studie gezielt zur Verankerung agiler Prinzipien beitragen. Die genannte Best Practice schafft dafür operative Freiräume und Lernmöglichkeiten, z. B. bei Einführung neuer Technologien oder Methoden, Umsetzungen für Zertifikate (z.B. Scrum Master, etc.)

Tabelle 27: Beziehung von CoPs und den Best Practices

6.6 Validität und Limitationen

In diesem Kapitel werden die zentralen methodischen und inhaltlichen Einschränkungen der vorliegenden Fallstudienforschung reflektiert. Dafür werden im Kapitel 6.6.1 die vier Validitätskriterien nach Yin [19] herangezogen. Anschließend wird im Kapitel 6.6.2 erklärt, inwieweit die Ergebnisse auf andere Projekte übertragbar sind und welche Grenzen dabei zu beachten sind. Ziel ist es, die Aussagekraft und die Einschränkungen der Erkenntnisse offen darzustellen.

6.6.1 Methodische Validität

Die Qualität qualitativer Fallstudien lässt sich nach Yin anhand von vier zentralen Validitätskriterien beurteilen: Konstruktvalidität, interne Validität, externe Validität und Reliabilität [19]. Diese ermöglichen eine strukturierte Einschätzung der methodischen Fundierung sowie möglicher Einschränkungen der Studie.

- **Konstruktvalidität (Engl. „Construct Validity“)**
  Dieses Kriterium prüft, ob die theoretischen Konzepte, die untersucht werden sollen – wie z. B. Selbstorganisation oder Best Practices – auch wirklich korrekt und verständlich im Forschungsprozess erfasst wurden.

  In dieser Arbeit wurde darauf geachtet, dass alle zentralen Begriffe klar definiert und in den Interviews einheitlich verwendet wurden. Durch semi-strukturierte Interviews mit offenen Fragen konnten individuelle Sichtweisen eingebracht werden, ohne die Befragten in eine bestimmte Richtung zu lenken (Vermeidung von Bias). Zusätzlich wurden weitere Datenquellen wie Task-Boards, Dokumente und Beobachtungen herangezogen (Triangulation), um Aussagen zu überprüfen und zu ergänzen.

Einschränkung: Begriffe könnten – abhängig von Rolle, Projektkontext, Erfahrungsniveau oder konkreter Umsetzung – unterschiedlich verstanden oder gewichtet worden sein.

- **Interne Validität (Engl. „Internal Validity“)**
  Dieses Kriterium bewertet, ob die Schlussfolgerungen inhaltlich plausibel aus den erhobenen Daten abgeleitet wurden.

  Auch wenn keine Ursache-Wirkung-Beziehungen im engeren Sinn untersucht wurden, wurde bei der Analyse darauf geachtet, wiederkehrende Muster zu erkennen und alternative Deutungen in die Interpretation einzubeziehen. Um Unklarheiten im Gespräch zu vermeiden, wurden bei Bedarf verständnissichernde Rückfragen gestellt, insbesondere wenn Begriffe oder Kontexte uneindeutig waren.

  Einschränkung: Abweichende Aussagen wurden zwar in den Transkripten erfasst, jedoch im Rahmen der Ergebnisdarstellung nicht gesondert ausgewertet. Zudem bleibt qualitative Analyse grundsätzlich interpretationsabhängig, eine vollständige Neutralität ist nicht möglich.

- **Externe Validität (Engl. „External Validity“)**
  Die externe Validität betrifft die Frage, ob die Ergebnisse auf andere, vergleichbare Kontexte übertragbar sind.

  In dieser Arbeit wurden drei Softwaregroßprojekte untersucht, die hinsichtlich Teamgröße, Arbeitsweise (z. B. Scrum, Kanban), technischer Infrastruktur und Organisationsstruktur ähnlich aufgebaut sind. Die Interviews folgten einem gemeinsamen, thematisch abgestimmten Leitfaden, wodurch inhaltlich vergleichbare Aussagen erhoben werden konnten. Die daraus entwickelten Best Practices erscheinen unter vergleichbaren Bedingungen übertragbar und bieten eine praxisnahe Orientierung für ähnlich aufgebaute Softwaregroßprojekte.

  Einschränkung: Für deutlich abweichende Kontexte – etwa kleinere Organisationen, andere Branchen oder klassische Entwicklungsmethoden – ist eine direkte Übernahme nicht immer möglich. Einige der abgeleiteten Best Practices zeigen jedoch Skalierungspotenzial und könnten mit entsprechender Anpassung auch in kleineren oder anders strukturierten Projekten wirksam sein.

- **Reliabilität (Engl. „Reliability“)**
  Das methodische Vorgehen – von der Datenerhebung bis zur Analyse – wurde vollständig dokumentiert. Die Interviews wurden transkribiert, mithilfe von MAXQDA softwaregestützt codiert und nach dem etablierten Verfahren der thematischen Analyse ausgewertet. Alle Arbeitsschritte wurden konsistent auf alle drei Fälle angewendet, wodurch ein hohes Maß an Nachvollziehbarkeit gewährleistet ist.
  Einschränkung: Qualitative Forschung ist grundsätzlich durch subjektive Deutungen geprägt. Auch bei sorgfältiger Durchführung lässt sich eine vollständige Wiederholbarkeit durch andere Forschende nicht garantieren – wohl aber eine methodische Nachvollziehbarkeit.

6.6.2 Ergebnisbezogene Limitationen und Übertragbarkeit

Die Ergebnisse dieser Arbeit basieren auf der qualitativen Analyse von drei Fallstudien in unterschiedlich ausgerichteten, aber strukturell vergleichbaren Softwaregroßprojekten. Dabei wurden ausschließlich jene Themen und Codes der thematischen Analyse weiterverwendet, die in allen drei Fällen übereinstimmend identifiziert wurden. Dies stärkt die Relevanz der daraus abgeleiteten Best Practices, schränkt jedoch die Breite potenzieller Erkenntnisse ein – etwa solche, die nur in einem oder zwei Fällen aufgetreten sind.

Die Auswahl der Themen basiert auf offenen semistrukturierten Interviewfragen, wodurch sich die Tiefe und Perspektive der Antworten stark an der Wahrnehmung der Interviewpartner orientiert. Es ist daher möglich, dass einzelne relevante Aspekte nicht benannt oder im Gesprächsverlauf nicht ausführlich angesprochen wurden. Ergänzende Artefakte und Beobachtungen aus der Participant Observation konnten dies nur teilweise kompensieren.

Zudem wurde eine begrenzte Anzahl an Projekten und Teamkonstellationen betrachtet. Die tatsächliche Wirksamkeit und Durchführbarkeit der Praktiken in der Praxis, hängt stets von konkreten Faktoren wie Unternehmenskultur, Teamdynamik, Rollenverständnis oder technologischem Reifegrad ab. Eine pauschale Übertragbarkeit kann daher nicht angenommen werden.

In den Fallstudien wurde eine breite Rollendiversität berücksichtigt. Allerdings wurde pro Fallbeispiel meist nur eine Person je Rolle befragt, teils mit Mehrfachrollen. Dadurch konnten unterschiedliche Perspektiven abgebildet, aber keine Tiefe innerhalb einzelner Rollenprofile im jeweiligen Projektkontext erreicht werden. Eine erweiterte Stichprobe könnte zusätzliche Einsichten und potenzielle Rollenkontraste innerhalb eines Teams ermöglichen.

Die gewählte Methodik – insbesondere die thematische Analyse nach Braun & Clarke – ermöglicht eine strukturierte, theoriebasierte Auswertung qualitativer Daten. Dennoch bleibt ein Rest an Subjektivität in der Interpretation bestehen.

7 Zusammenfassung und Ausblick

Diese Arbeit befasste sich mit der Frage, welche Praktiken für autonome Teams auch in industriellen Softwaregroßprojekten funktionieren und sich skalieren lassen. Ziel war es, aus realen Projekten konkrete Best Practices zu identifizieren, diese systematisch zu kategorisieren und Kriterien für ihre Einordnung zu entwickeln.

Als Grundlage wurde der aktuelle Stand der Forschung zu autonomen Teams analysiert. Die Kategorien und Praktiken von Hoda et al. wurden als strukturierender Ausgangspunkt gewählt. Ihre sieben Kategorien aus den Balancing Acts sind klar abgegrenzt und decken zentrale Bereiche wie Entscheidungsfindung, Verteilung von Aufgaben, Fortschrittsüberwachung und Wissensverteilung ab. Diese Struktur diente sowohl der Entwicklung des Interviewleitfadens als auch dem späteren Vergleich mit den empirischen Ergebnissen.

Zur Datenerhebung wurden drei Fallbeispiele aus unterschiedlichen Branchen untersucht:

- Fallbeispiel 1 – Versicherungsträger (inkl. Participant Observation)
- Fallbeispiel 2 – Autohaus
- Fallbeispiel 3 – Gesundheitsbereich

In jedem Fallbeispiel wurden fünf semistrukturierte Interviews mit Teammitgliedern aus verschiedenen Rollen durchgeführt. Dazu zählten Entwickler, Architekten, Requirements Engineers, Teamleiter und Projektleiter. Die Interviews erfolgten in semi-strukturierter Form, wurden aufgezeichnet, transkribiert und anschließend in MAXQDA, einer Analysesoftware, kodiert. Die Codierung erfolgte in mehreren Schritten und bezog sich direkt auf konkrete Aussagen und deren Kontexte.

Im Rahmen der Participant Observation im Fallbeispiel 1 wurden zusätzlich projektinterne Artefakte wie Kanban-Boards, Ticket-Systeme oder Dokumentationen analysiert, um die Aussagen aus den Interviews besser einordnen zu können. Diese ergänzende Perspektive war in den beiden anderen Fallbeispielen nicht möglich, da dort kein Zugang zu den Systemen bestand.

Die Auswertung erfolgte nach der thematischen Analyse nach Braun & Clarke. Zunächst wurden aus den codierten Passagen Themen abgeleitet und anschließend zu 13 übergeordneten Kategorien zusammengefasst, die zentrale Aspekte der Teamarbeit in großen Softwareprojekten abbilden. Beispiele dafür sind Transparenz, Kommunikation, Meetings, Wissensverteilung oder Aufwandsschätzung.

Aus den codierten Inhalten wurden insgesamt 23 konkrete Best Practices abgeleitet. Für das finale Ergebnis wurden ausschließlich jene Themen und Codes berücksichtigt, die in allen drei Fallbeispielen vorkamen. Diese gemeinsame Schnittmenge wurde bewusst gewählt, um nur solche Praktiken zu berücksichtigen, die eine möglichst hohe Aussagekraft und Relevanz besitzen. Kodierungen die nur in einem oder zwei Fallbeispielen vorkamen, wurden zwar dokumentiert, aber nicht weiter analysiert. Diese könnten jedoch in zukünftigen Arbeiten berücksichtigt werden.

Ein selbst entwickeltes Java-Programm unterstützte die Auswertung der Codes, indem es Überschneidungen, Segmenthäufigkeiten und Zuweisungen zu Kategorien automatisiert analysierte. Damit konnte die Analyse systematischer und nachvollziehbarer durchgeführt werden.

Die drei Forschungsfragen wurden wie folgt beantwortet:

RQ1a – Welche Praktiken lassen sich in Softwaregroßprojekten zur Unterstützung autonomer Teams identifizieren?
Es wurden 23 praxistaugliche Best Practices identifiziert, die zentrale Handlungsfelder autonomer Teamarbeit in Softwaregroßprojekten abdecken.

RQ1b – Welche Praktiken für autonome Teams sind auch für große Softwareprojekte skalierbar?
Neun der von Hoda et al. beschriebenen Best Practices konnten in den Fallstudien inhaltlich bestätigt werden. Sie wurden jeweils durch eine kontextbezogene Best Practice in den Softwaregroßprojekten abgebildet und gelten somit als skalierbar.

RQ2 – In welche Kategorien können die identifizierten Praktiken unterteilt werden?
Die 23 Best Practices wurden 13 thematischen Kategorien zugeordnet, die zentrale Aspekte der Zusammenarbeit in autonomen Teams abbilden. Mehrere Praktiken wirken bereichsübergreifend und lassen sich mehreren Kategorien gleichzeitig zuordnen.

RQ3 – Auf Basis welcher Kriterien kann eine Kategorisierung stattfinden?
Die Kategorisierung der Best Practices basiert auf ihrer inhaltlichen Relevanz für die jeweilige Kategorie, auf der empirischen Ableitung aus den Codes der thematischen Analyse sowie auf der

thematischen Konsistenz innerhalb der Kategorien. Entscheidend war, ob eine Best Practice zur Zielsetzung und Wirkung einer Kategorie beiträgt, ob sie durch codierte Interviewaussagen inhaltlich gestützt wird, und ob sie sich in das Gesamtbild der jeweiligen Kategorie sinnvoll einfügt, ohne inhaltliche Widersprüche zu erzeugen.

7.1 Ausblick und zukünftige Forschung

Ausgehend von den vorliegenden Ergebnissen ergeben sich mehrere relevante Ansätze für weiterführende Forschung und praktische Anwendung.

- **Erprobung und langfristige Evaluierung in realen Projekten:**
  Die als skalierbar identifizierten Best Practices könnten gezielt in neuen Projekten eingesetzt werden, um ihre Alltagstauglichkeit in Bezug auf Teamgröße, Rollenverteilung und organisatorisches Umfeld zu überprüfen. Ergänzend dazu ließe sich ihre nachhaltige Wirksamkeit im Rahmen von Langzeitstudien über mehrere Projektzyklen hinweg beobachten und evaluieren. Auf diese Weise könnten sowohl kurzfristige Anpassungsbedarfe als auch langfristige Effekte identifiziert werden.

- **Quantitative Validierung:**
  Auf Grundlage der 23 identifizierten Praktiken könnten standardisierte Umfragen entwickelt werden, um deren Verbreitung, Relevanz und Wirksamkeit in unterschiedlichen Projektkontexten empirisch zu erfassen.

- **Vertiefung der Analyse nicht skalierbarer Praktiken:**
  Einige der in Kapitel 6.2.10 als nicht skalierbar klassifizierten Praktiken könnten in zukünftigen Studien nochmals gezielt untersucht werden. So zum Beispiel beim Best Practice „Multiple Perspectives“ (siehe Punkt 6.2.10.1) denkbar, da durch Mehrfachrollen zwar funktionale Überschneidungen existieren, der aktive Perspektivenaustausch jedoch nicht explizit benannt wurde.

- **Untersuchung fallstudien-spezifischer oder neuer Praktiken:**
  Themen und Codes, die nur in zwei der untersuchten Fallbeispiele auftraten, wurden in Kapitel 5.2.5 dokumentiert, aber nicht in das finale Kategoriensystem überführt. Für diese Codes erfolgte keine Ableitung konkreter Best Practices, da sie nicht in allen drei Fallbeispielen beobachtet wurden. Zukünftige Arbeiten könnten gezielt an diesen Schlagwörtern und Codierungen ansetzen – etwa durch vertiefende Interviews mit weiteren Teammitgliedern der bereits untersuchten Organisationen oder durch zusätzliche Fallstudien. Auf diese Weise ließe

sich klären, ob bestimmte Themen lediglich kontextbedingt nicht erwähnt wurden oder ob sie tatsächlich projektspezifisch sind.

- **Übertragung auf kleinere Softwareprojekte:**
  Einige der neu identifizierten Praktiken könnten auch in kleineren autonomen Teams eingesetzt werden, um im Gegenzug zu prüfen, ob sie auch in kleinen Softwareprojekten funktionieren.

- **Integration von KI-Werkzeugen:**
  Mit Blick auf aktuelle Entwicklungen in der Softwarebranche stellt sich die Frage, inwiefern KI-basierte Tools die Umsetzung bestimmter Best Practices unterstützen oder verbessern können – etwa in den Bereichen Dokumentation, Wissensverteilung oder Review-Prozesse. Erste Ansätze dazu finden sich bereits in der Literatur (vgl. [58]).

Die in dieser Arbeit gewonnenen Ergebnisse können als fundierte Grundlage für weiterführende Forschung dienen und Unternehmen erste Anhaltspunkte bieten, wie autonome Teams auch in komplexen Softwareprojekten gezielt gefördert und begleitet werden können.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] S. S. Maidin und N. Yahya, „The Waterfall Model with Agile Scrum as the Hybrid Agile Model for the Software Engineering Team," Yogyakarta Indonesia, IEEE, 2022.

[2] N. B. Moe, V. Stray und R. Hoda, „Trends and Updated Research Agenda for Autonomous Agile Teams: A Summary of the Second International Workshop at XP2019," in *Agile Processes in Software Engineering and Extreme Programming -- Workshops*, Springer International Publishing, 2019, pp. 13-19.

[3] K. Dikert, M. Paasivaara und C. Lassenius, „Challenges and success factors for large-scale agile transformations: A systematic literature review," Journal of Systems and Software, 2016, pp. 87-108.

[4] N. B. Moe und V. Stray, „A Decade of Research on Autonomous Agile Teams: A Summary of the Third International Workshop," Trondheim, Norway, Springer, 2020.

[5] N. B. Moe, D. Bjørn, V. Stray, L. S. Karlsen und S. Schjødt-Osmo, „Team Autonomy in Large-Scale Agile," 2019.

[6] M. Berntzen, R. Hoda, N. B. Moe und V. Stray, „A Taxonomy of Inter-Team Coordination Mechanisms in Large-Scale Agile," IEEE, 2022.

[7] S. M. Sablis, „Team-external coordination in large-scale software development projects," Trondheim Norway, Journal of Software: Evolution and Process, 2020.

[8] E. Bjarnason, B. G. Bern und L. Svedberg, „Inter-team communication in large-scale co-located software engineering: a case study," Springer, 2022.

[14] H. Edison, X. Wang und K. Conboy, „Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review," IEEE, 2022.

[16] S. Rahim, E. Chowdhury, D. Nandi, M. Rahman und S. Hakim, „ScrumFall: A Hybrid Software Process Model," International Journal of Information Technology and Computer Science (IJITCS), 2018, pp. 41-48.

[17] „Estimation of software quality parameters for hybrid agile process model," Springer SN Applied Sciences, 2021, pp. 1-11.

[18] M. Fowler und J. Highsmith, „The Agile Manifesto," Agile Alliance, 2001.

[19] R. K. Yin, „Case Study Research and Applications: Design and Methods," 6 Hrsg., I. SAGE Publications, Hrsg., 2017.

[20] P. Runeson und M. Höst, „Guidelines for conducting and reporting case study research in software engineering," in *Empirical Softw. Engg.*, 14 Hrsg., 2009, p. 131–164.

[21] K. W. Grümer, Techniken der Datensammlung 2, 1 Hrsg., Bd. 32, Wiesbaden: Vieweg+Teubner Verlag, 1974.

[22] W. Chauncey, „Interview Techniques for UX Practitioners," Boston, Morgan Kaufmann, 2014, pp. 23-41.

[23] V. Braun und V. Clarke, Using thematic analysis in psychology. Qualitative Research in Psychology, 2006, pp. 77-101.

[24] R. Hoda, „Self-Organizing Agile Teams: A Grounded Theory," Victoria University of Wellington, 2011.

[25] R. Hoda, J. Noble und S. Marshall, „Balancing Acts: Walking the Agile Tightrope," ACM, Cape Town, South Africa, 2010.

[26] R. Hoda, J. Noble und S. Marshall, „Developing a grounded theory to explain the practices of self-organizing Agile teams," Springer Sciencd+Business Media, 2012.

[27] M. Dr. Morner, Organisation der Innovation im Konzern, Springer Fachmedien Wiesbaden, 1997, p. 143ff..

[28] J. W. Cresswell, „Qualitative Inquiry & Research Design - Choosing Among Five Approaches," 3 Hrsg., Thousend Oaks, California, SAGE Publications Inc., 2013, p. 97.

[29] J. Gustaffson, „Single case studies vs. multiple case studies: A comparative study," Halmstad University Sweden, 2017.

[30] K. Fisher, „Leading Self-Directed Work Teams: A Guide to Developing New Team Leadership Skills," Mcgraw Hill Book Co, 2000, p. 17.

[31] L. Yunbo, L. Lan, W. Hongli, L. Yun und S. Qian, „Measurement model of project complexity for large-scale," in *International Journal of Project Management*, Bd. 33, Shanghai, Elsevier Ltd. APM and IPMA, 2015, pp. 610-622.

[32] D. Baccarini, „The concept of project complexity - a review," in *International Journal of Project Management*, Bd. 14, Perth, Australia, Elsevier Science Ltd and IPMA, 1996, pp. 201-204.

[33] J. Pokorny, „NoSQL databases: a step to database scalability in web environment," in *International Journal of Web Information Systems*, 9 Hrsg., Bd. 1, Charles University, Praha, Czech Republic, 2013, pp. 69-82.

[35] V. Mahnic und S. Drnovscek, „Agile Software Project Management with Scrum," University of Ljubljana, Slovenia, 2005.

[36] R. Dräther, H. Koschek und C. Sahling, „Scrum-kurz & gut 3. Auflage," O'Reilly, 2023, p. 22ff..

[37] K. Beck und C. Andres, „Extreme Programming Explained," 2. Edition Hrsg., Addison-Wesley Professional, 2004.

[39] L. Lindstrom und R. Jeffries, „Extreme Programming and Agile Software Development Methodologies," in *Information Systems Management*, 2004, p. 43ff.

[42] „Extreme Programming Vs Scrum: A Comparison Of Agile Models.," International Journal of Technology, Innovation and Management (IJTIM) Vol.2, Issue.2, 2022, pp. 80-96.

[44] E. Weflen, C. A. MacKenzie und I. V. Rivero, „An influence diagram approach to automating lead time estimation in Agile Kanban project management," in *Expert Systems with Applications 187*, 2022.

[45] E. Mircea, „Project Management using Agile Frameworks," Academy of Economic Studies. Economy Informatics Vol. 19, 2019, p. 34 ff..

[47] H. Takeuchi und I. Nonaka, The New New Product Development Game, Harvard Business School, 1986.

[48] R. Hoda, J. Noble und S. Marshall, „Organizing Self-Organizing Teams," ACM, Cape Town, South Africa, 2010.

[49] M. Kalenda, P. Hyna und B. Rossi, „Scaling Agile in Large Organizations: Practices, Challenges and Success Factors," Journal of Software: Evolution and Process, 2018.

[50] V. Stray, N. B. Moe und R. Hoda, „Autonomous Agile Teams: Challenges and Future Directions for Research," Porto, Portugal, Association for Computing Machinery, 2018.

[51] Z. Masood, R. Hoda und K. Blincoe, „What Drives and Sustains Self-Assignment in Agile Teams," IEEE Transactions on Software Engineering, 2022, pp. 3626-3639.

[52] Z. Masood, R. Hoda und K. Blincoe, „How agile teams make self-assignment work: a grounded theory study," Empirical Software Engineering 25, 2020.

[53] M. G. Rothstein und R. J. Burke, Self-Management and Leadership Development, Cheltenham, UK • Northampton, MA, USA: Edward Elgar Publishing Limited, 2010, p. 271ff..

[54] E. Salas, M. A. Rosen, S. C. Burke und G. F. Goodwin, „The wisdom of collectives in organizations: An update of the teamwork competencies.," New York, Routledge/Taylor & Francis Group, 2009, pp. 39-79.

[56] C. Bremer, A. R. Eklund und M. Elmquist, „Scaling or growing agile? Proposing a manifesto for agile organization development," *Journal of Organization Design,* pp. 23-24, 2025.

[57] F. Tobisch, J. Schmidt und F. Matthes, „Investigating Communities of Practice in Large-Scale Agile Software Development: An Interview Study," in *Agile Processes in Software Engineering and Extreme Programming*, Munich, Germany, Springer, 2024, pp. 3-19.

Weblinks

[9] „SAFe,“ [Online]. Available: https://framework.scaledagile.com/. [Zugriff am 24 03 2025].

[10] „less,“ [Online]. Available: https://less.works/. [Zugriff am 18 04 2025].

[11] „scrum@scale,“ [Online]. Available: https://www.scrumatscale.com/. [Zugriff am 18 04 2025].

[12] „Disciplined Agile® Delivery,“ [Online]. Available: https://www.pmi.org/disciplined-agile. [Zugriff am 18 04 2025].

[13] „Atlassian,“ [Online]. Available: https://www.atlassian.com/agile/agile-at-scale/spotify. [Zugriff am 18 04 2025].

[15] „Monash University,“ [Online]. Available: https://research.monash.edu/en/persons/rashina-hoda/publications. [Zugriff am 01 04 2025].

[34] „scrumalliance,“ [Online]. Available: https://www.scrumalliance.org/about-scrum. [Zugriff am 18 05 2025].

[38] „Agile Alliance,“ [Online]. Available: https://www.agilealliance.org/glossary/xp/. [Zugriff am 18 05 2025]

[40] „extremeprogramming.org,“ [Online]. Available: http://www.extremeprogramming.org/values.html. [Zugriff am 18 05 2025].

[41] [Online]. Available: https://www.c-sharpcorner.com/article/12-core-practices-in-xp/. [Zugriff am 15 1 2025].

[43] [Online]. Available: https://www.praxisframework.org/files/royce1970.pdf. [Zugriff am 05 05 2025].

[46] „ionos.at,“ [Online]. Available: https://www.ionos.at/digitalguide/websites/web-entwicklung/kanban/. [Zugriff am 18 05 2025].

[55] „MAXQDA,“ [Online]. Available: https://www.maxqda.com/. [Zugriff am 18 05 2025].

[58] R. Dagley, „ITProToday,“ 15 01 2025. [Online]. Available: https://www.itprotoday.com/software-development/software-development-trends-and-predictions-2025-from-industry-insiders. [Zugriff am 24 03 2025].

Anhang

Übersicht verwendeter Hilfsmittel

Tool	Version / Datum	Verwendungsbereich	Beispielhafte Eingaben (Prompts)	Verwendung ohne substanzielle Änderungen?
ChatGPT (OpenAI)	GPT-4, Jan–März 2025	Hilfe bei Formulierung und Strukturvorschlägen von Kapiteltexten bei 6.1.	„Reformuliere diese Beschreibung für diese Praktik basierend auf folgenden Text…“	Nein – alle Texte wurden überarbeitet und angepasst.
	GPT-4, März 2025	Tabelle 26 und Tabelle 27	„Erstelle aus dem Cop Zielen … und relevanten Best Practices … wobei die Zuordnung wie folgt aussieht … eine Tabelle.“	Begründungen bzw. Einordnung und Formatierung adaptiert.
	GPT-4, Jan–März 2025	Konsistenzprüfung von Kapitelnummern und Bezeichnungen	„Überprüfe ob alle Themen/Kategorien in Kapitel 6 mit demselben Wortlaut konsistent verwendet wurden“.	Nein, wurde zur Überprüfung gleicher Formulierungen zwecks Konsistenz verwendet.
	GPT-4, März 2025	Kurzfassung/Abstract	„Erstelle einen Vorschlag für die Kurzfassung basierend auf dem Inhalt dieser Arbeit“.	Nein, Textvorschlag wurde adaptiert und geändert.

Tabelle 28: Übersicht verwendeter Hilfsmittel

Übersicht Methodik mit den dazugehörigen Kapiteln

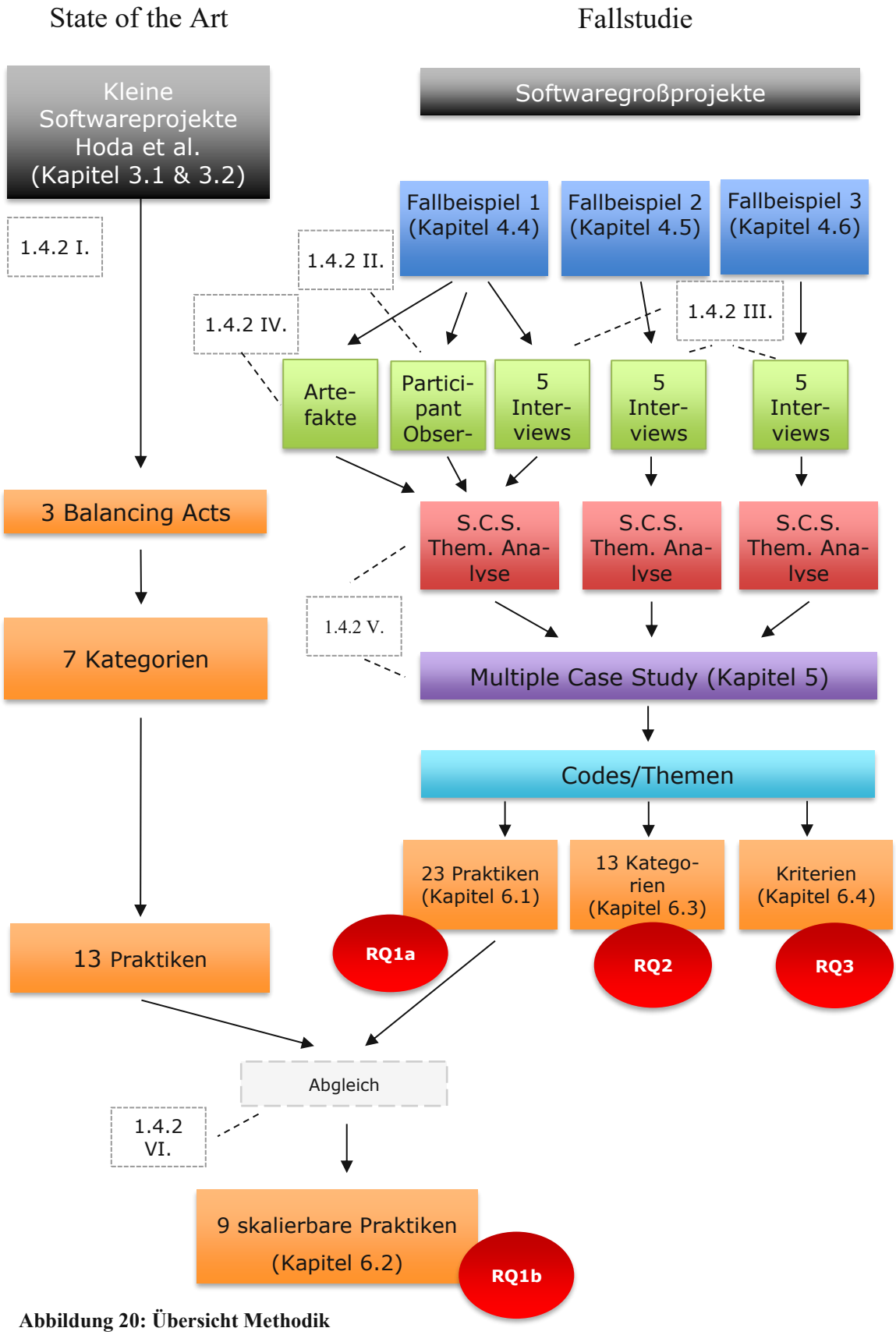


Abbildung 20: Übersicht Methodik

Interviewleitfaden

Allgemeine Information

Datum:

Teilnehmer:

Thema: Arbeitsweise autonomer Teams in Softwaregroßprojekten.

Ziel dieses Interviews ist es die Arbeitsweise autonomer Teams in Softwaregroßprojekten zu erfassen.

Regeln

- Das Interview wird per Aufnahmegerät und der Einwilligung des Interviewteilnehmers aufgenommen damit der Interviewer nicht alle Informationen während des Interviews schriftlich festhalten muss und eine leichtere Auswertung ermöglicht wird.
- Der Interviewte Teilnehmer darf jederzeit auch relevante Themen diskutieren die nicht unmittelbar durch die Interviewfragen abgedeckt sind.

Interview-Fragen

Allgemeines:

1. Was ist das Ziel des Projektes und welchem Thema sind Sie zugeteilt?
2. Welche Rolle und welchem Verantwortungsbereich sind Sie zugeteilt?
3. Wie viele Teilnehmer gibt es in dem Projekt und in Ihrem autonomen Team?
4. Wie sieht die Projekt- und Hierarchie-Struktur aus?
   (Releasezyklen, Teilprojekte und Hierarchie)
5. Erklären Sie den eingesetzten Softwareentwicklungsprozess und dessen Anwendung?
   (Traditionelles oder agiles Vorgehensmodell)

Kategorie 1: Collective decision making

6. Wie und von wem werden Komplexität- und Aufwandsschätzungen von Aufgaben gemacht?
7. Wie und von wem werden Richtlinien und Prinzipien für die Umsetzung festgelegt?
8. Wie wird auf dynamische Änderungen bei Anforderungen fachlicher oder technischer Hinsicht umgegangen?

Kategorie 2: Self-assignment

9. Wie erfolgen die Auswahl und Zuteilung von Anforderungen und Fehlern?
10. Wie wird die Transparenz für alle Teammitglieder gewährleistet?

Kategorie 3: Self-monitoring

11. Wie und von wem erfolgt die Fortschrittsmessung der Aufgaben?
12. Gibt es Meetings um den Fortschritt anderen Teilnehmern/Teams zu kommunizieren und Probleme rechtzeitig zu erkennen und zu beheben?

Kategorie 4: Need for specialization

13. Besteht die Möglichkeit der individuellen Fortbildung innerhalb des Teams?
14. Besteht die Möglichkeit der individuellen über die Teamgrenzen hinaus?

(Andere Themengebiete, Rollenwechsel)

Kategorie 5: Encouraging cross-functionality

15. Wie erfolgt die Kommunikation innerhalb und außerhalb ihres autonomen Teams?
16. Wie erfolgt die Wissensverteilung innerhalb des Teams?

(Wissensverteilung, Spezialisierung und Support, Umgang mit Ausfällen)

Kategorie 6: Self-evaluation

17. Besteht zwischen den Iterationen eine Reflexion der individuellen und gesamten Teamleistung (Lernprozess)?
18. Gibt es einen Ansporn für gute Team- oder Individualleistungen?

Kategorie 7: Self-improvement

19. Gibt es Praktiken die eingesetzt werden um komplexe Aufgabengebiete zu bewältigen?
20. Wird den Teammitgliedern die Möglichkeit geboten neue Themengebiete zu erforschen oder zu vertiefen? (Fortbildungen, Kurse, etc.)

Zusatz:

21. Gibt es noch relevante Punkte welcher nicht durch die oben angeführten Interviewfragen abgedeckt wurde und der für das Arbeiten als autonomes Team für Sie wichtig ist?