MASTERARBEIT

ENTWICKLUNG VON STRUKTUREN ZUR ETABLIERUNG MODELLGETRIEBENER SOFTWAREENTWICKLUNG IN EINEM INTERNATIONAL VERTEILTEN TEAM

ausgeführt am

Studiengang

Informationstechnologien und Wirtschaftsinformatik

Von: Michael Kircher

Personenkennzeichen: 1810320009

Graz, am 15. Juni 2020 .....

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

Danken möchte ich meinem Betreuer Christian Hofer, Bakk, BSc, MSc für die gute Zusammenarbeit sowie das stets schnelle, freundliche und konstruktive Feedback während des Erstellungsprozesses dieser Arbeit.

Ein weiterer Dank gilt meiner Freundin Wella, welche mir stets liebevoll und hilfsbereit durch schwierige Phasen dieser Arbeit geholfen hat.

Außerdem bedanke ich mich bei meinen KollegInnen der ANDRITZ Gruppe – durch die vielen Gespräche konnte ich wertvolle Sichtweisen und Erfahrungen lukrieren, welche beim Erstellen dieser Arbeit enorm hilfreich waren.

Zum Schluss bedanke ich mich noch bei meiner Familie, welche mich stets auf allen Ebenen unterstützt.

KURZFASSUNG

Um den steigenden Bedarf an IT-ExpertInnen, welcher durch die Digitalisierung von Geschäftsprozessen verstärkt auch in Unternehmen zum Tragen kommt, deren Kerngeschäft nicht im Bereich IT oder Softwareentwicklung liegt, entgegenzuwirken, setzt die ANDRITZ Gruppe seit 2018 Mendix als Low-Code bzw. No-Code Plattform ein. Durch den Einsatz solcher Plattformen, mit denen Softwarelösungen von nicht IT-Experten direkt in der Fachabteilung umgesetzt werden können, ergeben sich jedoch neue Herausforderungen betreffend der Zusammenarbeit zwischen IT- und Fachabteilung.

Ziel dieser Arbeit ist es daher, Strukturen zur modellgetriebenen Softwareentwicklung in einem international verteilten Team, bestehend aus IT-ExpertInnen und MitarbeiterInnen der Fachabteilungen, zu etablieren, um Softwareprojekte effektiv über Abteilungsgrenzen hinweg umzusetzen.

Dazu wurden anhand einer Literaturrecherche entlang des Software-Lebenszyklus Einflussfaktoren identifiziert, welche die effektive Zusammenarbeit in einem dezentralen Team beeinflussen können. Um diesen Faktoren je nach Projektsituation individuell begegnen zu können, wurde ein eigener Servicebereich innerhalb der ANDRITZ IT etabliert welche sämtlichen Tätigkeiten rund um das Thema Rapid Application Development (RAD) steuert. Kern dieses Servicebereichs ist ein Center of Excellence (CoE), welches die zentrale Steuerungs- und Koordinationsaufgabe übernimmt und je nach Projekt entscheidet, welche Umsetzungsmethoden, Werkzeuge und Rollen für eine erfolgreiche Umsetzung benötigt werden.

Nach einer Einführung in die für diese Arbeit relevanten Theorietischen Grundlagen wird das ANDRITZ RAD CoE und dessen Aufgaben detailliert beschrieben. Im empirischen Teil dieser Arbeit wird der Einsatz des CoE an einem real durchgeführten Projekt innerhalb der ANDRITZ Gruppe getestet und anschließend mittels ExpertInneninterviews eruiert, ob sich die entwickelte Vorgehensweise für den praktischen Einsatz eignet oder ob Anpassungen durchgeführt werden müssen .

ABSTRACT

In order to manage the increasing demands on IT-Experts, brought about by the growing need for business-process digitalization in companies which are not IT focused, the ANDRITZ Group decided to introduce Mendix as a Low-Code / No-Code Software Development Platform at the end of 2018. The utilization of such Platforms, which allow non-IT experts to develop software solutions directly in the business-related departments, introduces new challenges due to the direct collaboration during the software creation process between IT and business.

The goal of this Master Thesis was to determine, which structures are necessary to establish model-driven software development in an internationally distributed team, consisting both of IT-Experts and non-IT Experts, in order to successfully finish software projects across department boundaries.

To this end, the influencing factors, that can affect collaboration in de-centralized teams were identified based on a literature review orientated on the Software Development Lifecycle. In order to deal with each factor individually depending on the type of project being implemented, a dedicated Service Area was then established within the ANDRITZ IT, which coordinates all activities related to Rapid Application Development (RAD). The heart of this Service Area is a Center of Excellence (CoE), which deals with both governance and project coordination, as well as deciding which set of processes, tools and roles are necessary for successful implementation of each individual project.

After establishing the theoretical background, the ANDRIZ RAD CoE is described in detail including all of its tasks and duties. In the empirical part of the thesis, the established structures were first tested in a real-world project (case study) - to collect data, Expert Interviews were then conducted to determine if the established structures are efficient or if adaptations are needed.

INHALTSVERZEICHNIS

1 EINLEITUNG

In Zeiten der Digitalisierung sind Unternehmen zunehmend mit dem Problem konfrontiert, eine Unmenge an historisch gewachsenen Geschäftsprozessen bzw. Aufgaben, welche derzeit in Form von zum Beispiel Excel- oder PDF-Dokumenten bearbeitet, umgesetzt und dokumentiert werden, als digitale cloudbasierte Lösungen zur Verfügung zu stellen. Einerseits ermöglichen solche digitalen Lösungen, seien es webbasierte mobile Applikationen für Smartphones bzw. Tablets oder responsive Websites, eine einfache und unkomplizierte Zusammenarbeit aller am Prozess beteiligten Personen – die Daten sind immer und überall verfügbar sowie stets aktuell - andererseits ist durch die zentrale Datenspeicherung deren Weiterverarbeitung zum Zwecke einer späteren Analyse gewährleistet.

Der Bedarf an qualifizierten Softwareentwicklern ist in den letzten Jahren enorm gestiegen, daher ist es für Unternehmen oft schwierig geeignetes Personal zu finden um zeitnah, neben bereits laufenden Projekten auf die oben genannten Anforderungen zu reagieren. Das zeigt auch ein Blick in den Digital Dossier des Bundesministeriums für Digitalisierung und Wirtschaftsstandort (2018), welcher einen erheblichen Bedarf an IT-SpezialistInnen in Österreich feststellt: Obwohl österreichische Unternehmen im EU-weiten Vergleich auf einen überdurchschnittlich hohen Anteil an IT-Fachkräften von rund vier Prozent der Beschäftigten zurückgreifen können (mit leicht steigender Tendenz), ist die Nachfrage nach diesen Spezialisten mit rund 67 Prozent jedoch auch höher als im europaweiten Durchschnitt welcher bei etwa 48 Prozent liegt (Bundesministerium für Digitalisierung und Wirtschaftsstandort, 2018).

Um diesen Engpass entgegenzuwirken, bieten sich Werkzeuge zur modellgetriebenen Softwareentwicklung, genauer gesagt Low-Code bzw. No-Code Entwicklungsplattformen an. Solche Plattformen ermöglichen es, Geschäftslogik durch visuelle Modellierung unter Zuhilfenahme vorgefertigter Komponenten, ähnlich der Business Process Modeling Language (BPML), zu erstellen und die relevanten Daten in einer mittels „What You See Is What You Get“ (WYSIWYG) kreierten Benutzeroberfläche zu präsentieren. Dadurch können auch MitarbeiterInnen der Fachabteilungen, welche wenig bzw. keine Entwicklungserfahrung aufweisen, nach vergleichsweise kurzer Einarbeitungszeit eigenständig Anwendungen erstellen und als so genannte Citizen Developer fungieren. Damit diese direkt in den Fachabteilungen stattfindenden Entwicklungen jedoch nicht zu einer so genannten „Schatten IT“ führen, muss die IT-Abteilung Prozesse, Regeln und Standards definieren sowie bis zu einem gewissen Grad aktiv in den Entwicklungsprozess eingebunden werden.

Das Unternehmen ANDRITZ AG nutzt zu diesem Zweck seit Anfang 2018 Mendix als Low-Code Entwicklungsplattform. Derzeit wird Mendix ausschließlich zentral in der IT-Abteilung von EntwicklerInnen, welche bereits mehrjährige Erfahrung in verschiedenen objektorientieren Programmiersprachen aufweisen, benutzt. Ziel ist es jedoch, dieses Werkzeug auch MitarbeiterInnen in den Fachabteilungen, die bislang wenig beziehungsweise keine Erfahrung mit Softwareentwicklung haben, zugänglich zu machen. Somit können kleinere Digitalisierungsprojekte innerhalb der Fachabteilung mit minimaler Unterstützung der IT umgesetzt werden. Der Betrieb der mittels dieser Plattform erzeugten Anwendungen in der Cloud oder On-Premises [1], wird dabei jedoch weiterhin zentral von der IT verwaltet.

1.1 Zielsetzung & Forschungsfrage

Das Ziel dieser Arbeit ist es zu eruieren, welche Regeln, Strukturen und Prozesse nötig sind, um in einem global verteilten Team bestehend aus IT-ExpertInnen und Citizen Developern einen effizienten Entwicklungsprozess zu gewährleisten und Entwicklungen außerhalb des Einflussbereiches der IT zu beschränken. Im Fokus steht dabei der gesamte Software-Lebenszyklus, von der ersten Idee und dem daraus resultierenden Anforderungsmanagement, über die Entwicklung der Anwendung und deren Qualitätskontrolle bis hin zum Go-Live sowie Betrieb der Anwendung und der darüber hinaus andauernden Wartung. Folgende Fragestellungen sollen mit dieser Arbeit beantwortet werden: Welche Informationen müssen in welcher Form verfügbar sein? Welche Arbeitsschritte können vom Citizen Developer in der Fachabteilung selbst durchgeführt werden und ab wann müssen ExpertInnen der IT-Abteilung in diesen Prozess mit eingebunden werden? Welche Rollen und Verantwortlichkeiten ergeben sich aus der Zusammenarbeit zwischen Fach- und IT-Abteilung? Welche Schulungsmaßnahmen sind notwendig damit Personen aus den Fachbereichen komplexere Anwendungen umsetzen können? Aus diesen Einzelfragen lässt sich die zentrale Forschungsfrage dieser Masterarbeit ableiten, welche wie folgt lautet:

Welche zentralen Einflussfaktoren sind zu berücksichtigen, wenn Strukturen zur modellgetriebenen Softwareentwicklung mittels Mendix in einem international verteilten Team eingeführt werden, welches über begrenztes Wissen bezüglich Softwareentwicklung verfügt?

[1] Bei on-Premises Lösungen wird die Software im Gegensatz zu Cloud-Lösungen auf einem Server im unternehmenseigenen Netzwerk betrieben.

Zusätzlich wurden folgende Hypothesen erstellt:

H1: Anhand der entwickelten Strukturen ist modellgetriebene Softwareentwicklung in einem international Verteilten Team, welches über begrenztes Wissen in diesem Bereich verfügt, möglich.

H0: Anhand der entwickelten Strukturen ist modellgetriebene Softwareentwicklung in einem international Verteilten Team, welches über begrenztes Wissen in diesem Bereich verfügt, nicht möglich.

H2: Durch vorab definierte und einfach zugängliche Standards & Best Practices wird der Entwicklungsprozess für Nicht IT-ExpertInnen vereinfacht und hat dadurch einen positiven Einfluss auf den Projekterfolg.

H3: Durch das Definieren von Kommunikationskanälen und -regeln ist eine zielgerichtete und effektive Kommunikation innerhalb des Projektteams gewährleistet und hat dadurch einen positiven Einfluss auf den Projekterfolg.

1.2 Aufbau der Arbeit

Im ersten Teil dieser Arbeit wird der theoretische Hintergrund der vorliegenden Thematik anhand der Erkenntnisse, welche während der Literaturrecherche gewonnen wurden, erläutert. Auf den theoretischen Teil dieser Arbeit folgt ein Methodenkapitel, in welchem die im Zuge dieser Arbeit angewandten wissenschaftlichen Methoden sowie deren theoretische Hintergründe beschreibt. Dies beinhaltet die Fallstudie sowie die durchgeführten ExpertInneninterviews. Darauf folgt das Kapitel Ergebnisse, in dem anhand des Software-Lebenszyklus Werkezuge, Methoden und Prozesse identifiziert werden, um die Zusammenarbeit zwischen IT-Abteilung und Citizen Developern steuern. Als Produkt dieser Masterarbeit ist ein Center of Excellence entstanden, welches in Kapitel Ergebnisse detailliert beschrieben ist. Nach einem Testprojekt innerhalb der ANDRITZ AG wurde anhand von ExpertInneninterviews eruiert, ob die gewählten Werkzeuge, Methoden und Prozesse in der Praxis bestehen oder ggf. Nachbesserungen stattfinden müssen. Die aus den ExpertInneninterviews gewonnenen Erkenntnisse werden Kapitel Ergebnisse im Anschluss an das vorgestellte Center of Excellence präsentiert. Im darauffolgenden Kapitel Diskussion wird beleuchtet, ob die Forschungsfrage beantwortet und die aufgestellten Hypothesen verifiziert werden konnten. Den Abschluss dieser Arbeit bildet das Kapitel Ausblick in dem erläutert wird, welche weiteren Schritte im Anschluss an diese Arbeit innerhalb der ANDRITZ AG gesetzt werden.

2 THEORIE

Ziel des nachfolgenden Abschnitts der Masterarbeit ist es, die Grundlagen der verwendeten Begrifflichkeiten zu erläutern, um eine Wissensbasis für den darauffolgenden Praxisteil zu schaffen. Folgende Themenschwerpunkte werden erläutert:

- Einführung in die Modellgetriebene Softwareentwicklung sowie Low-Code Plattformen und Beschreibung der innerhalb der ANDRITZ AG verwendeten Low-Code Plattform „Mendix“.
- Definition eines Vorgehensmodells im IT-Kontext sowie eine kurze Beschreibung des Rapid Application Development (RAD) Modells.
- Beschreibung des Software-Lebenszyklus sowie Erläuterungen zu den einzelnen darin enthaltenen Phasen.
- Eine kurze Einführung in die Themen DevOps sowie BizDevOps, welche die Zusammenarbeit über Abteilungsgrenzen hinweg beschreiben.
- Faktoren, welche bei der Zusammenarbeit in international Verteilten Teams, zu beachten sind.

2.1 Modellgetriebene Softwareentwicklung

Der Begriff der Abstraktion wird im Kontext der Softwareentwicklung sehr häufig verwendet und beschreibt eines der grundlegendsten und wichtigsten Prinzipien:

> *Unter Abstraktion versteht man Verallgemeinerung, das Absehen vom Besonderen und Einzelnen, das Loslösen vom Dinglichen. Abstraktion ist das Gegenteil von Konkretisierung.* (Balzert, 2011)

Abstrahieren bedeutet also, das Herausheben vom Wesentlichen bzw. das Beiseitelassen vom Unwesentlichen - oft wird anstelle von Abstraktion auch von Modellbildung gesprochen (Balzert, 2011). Modellgetriebene Softwareentwicklung (MDSD) ermöglicht es also, den Prozess der Anwendungsentwicklung auf einer höheren Abstraktionsebene durchzuführen als es bei der traditionellen Programmierung der Fall ist. Auf dieser Ebene kann der Quellcode, abhängig vom gewählten Ansatz, teilweise bzw. komplett automatisiert generiert werden. MDSD ist dabei ein Oberbegriff für Techniken, welche aus formalen Modellen lauffähige Software erzeugen (Stahl, Völter, & Efftinge, 2007).

Charakterisiert werden die verschiedenen Ansätze, indem die Art der Synchronisation zwischen Quellcode und Modell betrachtet wird. Das Spektrum reicht dabei von reinem Quellcode ohne separaten Modell, bis hin zu Modellen aus denen kein Quellcode erzeugt wird und welche nur zum Entwerfen dienen – eine Übersicht dieser Ausprägungen ist in Abbildung 1 ersichtlich und werden im Folgenden genauer beschrieben (Beydeda, Book, & Gruhn, 2005).

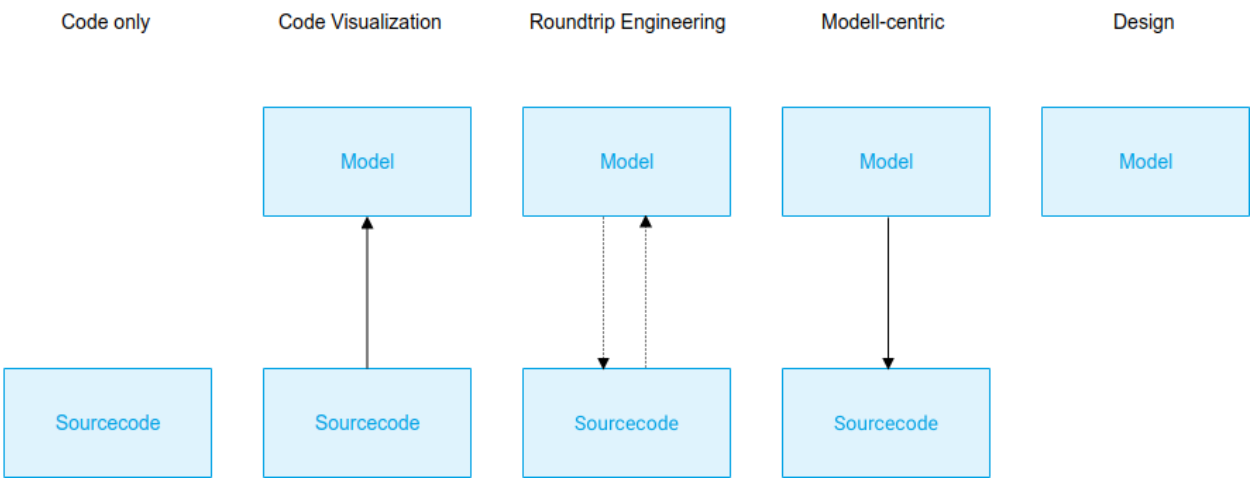

Abbildung 1: Modell - Code Interaktion in der Softwareentwicklung (eigene Darstellung in Anlehung an Beydeda, Book, & Gruhn, 2005)

- Code-only: Ist der am häufigsten verwendete Ansatz. Das „Modell“ ist dabei als Abstraktion direkt im Quellcode enthalten und wird durch Objekthierarchien (je nach gewählter Programmiersprache Packages, Module oder Interfaces) oder Bibliotheken repräsentiert. Der Prozess der Modellierung findet informal statt, zum Beispiel am Whiteboard oder im Kopf des Entwicklers (Wagner, 2014). Für kleine Projekte macht dieser Ansatz Sinn, jedoch verliert man bei größeren Projekten leicht den Überblick aufgrund der fehlenden grafischen Visualisierung der Zusammenhänge.
- Code-Visualization: Hilft dabei, die Struktur und das Verhalten der Software zu verstehen – Teile des Quellcodes werden dabei in einer gewählten Modellierungssprache (BPMN, UML, …) abgebildet (Wagner, 2014). Das Überführen von Modell in Quellcode findet jedoch weiterhin manuell statt. Es besteht also nur eine gedankliche Verbindung zwischen Modell und Code. Im Gegensatz zu Code-only, eignet sich dieser Ansatz um in großen Projekten, an denen teilweise mehrere Abteilungen involviert sind, den Überblick über die einzelnen Komponenten der Software und deren Funktionalität zu behalten.
- Round Trip Engineering (RTE): Hier findet eine wechselseitige Synchronisation zwischen Quellcode und Modell statt. Änderungen einer Seite werden automatisiert auf die andere Seite übertragen (Wagner, 2014). Dieser Ansatz sorgt für eine komplette Übereinstimmung zwischen Entwurf und Umsetzung. Ein Beispiel wäre hier das Erzeugen von Klassen und deren Methoden aus UML Diagrammen sowie das Reverse Engineering der Diagramme bei Änderungen im Quellcode. Modell und Code haben hier denselben Stellenwert. Im Unterschied zur Code-Visualisierung, fällt bei RTE die manuelle Komponente der Quellcodegenerierung weg. Das Abstraktionsniveau der Modelle ist bei diesem Verfahren jedoch dasselbe, wie beim Quellcode selbst (Stahl, Völter, & Efftinge, 2007).

- Model-centric: Bei diesem Ansatz ist das Modell das Hauptartefakt des Softwareentwicklungsprozesses (Wagner, 2014). Das Modell enthält alle nötigen Details - Beispielsweise eine Repräsentation der zu persistierenden / nicht persistierenden Daten, die komplette Geschäftslogik sowie das User Interface für die Präsentation der Daten - um daraus den kompletten Quellcode und eine lauffähige Anwendung zu generieren (Beydeda, Book, & Gruhn, 2005). Eine Weiterentwicklung bzw. Erweiterung dieses Ansatzes sind Low-Code Entwicklungsplattformen, welche in Kapitel 2.1.1 genauer betrachtet werden.
- Model-only: Hier werden die Modelle genutzt, um die Domäne bzw. den Lösungsraum einer Anwendung zu beschreiben und abzustecken. Sie dienen als Diskussionsgrundlage bzw. schaffen eine sprachliche Grundlage für alle am Projekt beteiligten Personen (Beydeda, Book, & Gruhn, 2005).

Die Motivation hinter der modellgetriebenen Softwareentwicklung, speziell der Vertreter der Model-centric Art ist es, den Fokus, welcher bei der traditionellen Entwicklung von Anwendungen zum großen Teil auf dem Schreiben des Quellcodes liegt, hin zur ganzheitlichen Modellierung der Lösung zu verlagern. Jede Software beinhaltet auf Quelltextebene Konstruktionsparadigmen bzw. eine innere Struktur, welche direkt die Entwicklungsgeschwindigkeit, Qualität, Performance, Wartbarkeit, Interoperabilität und Portabilität der Software beeinflusst. (Stahl, Völter, & Efftinge, 2007). Auf Quelltextebene ist diese Struktur jedoch nur noch schwer erkennbar, wodurch wertvolle Informationen verloren gehen. Als ganzheitliche Modellierung versteht man daher, die oft aufgrund von Zeit- oder Budgetgründen vernachlässigte Dokumentation (unter anderem durch Modelle), direkt mit dem Entwicklungsprozess zu verschmelzen – d.h. Modelle werden dabei nicht nur mehr als Dokumentation betrachtet sondern sind fester Bestandteil der Software (Stahl, Völter, & Efftinge, 2007).

Ein Vorteil modellgetriebener Ansätze besteht darin, dass Entwickler, die auf einer höheren Abstraktionsebene arbeiten, komplexere Systeme mit weniger Aufwand bewältigen können (Kleppe, Warmer, & Bast, 2003), was zu einer Steigerung der Produktivität führt. Außerdem bieten diese Ansätze Laien, durch die intuitive und leichter verständliche visuelle Modellierung, einen etwas einfacheren Einstieg in die Materie der Softwareentwicklung, indem man sich auf die zu implementierende Geschäftslogik konzentrieren kann, anstatt sich mit der Komplexität der zugrundeliegenden Programmiersprache auseinandersetzen zu müssen.

Daher ist es nicht überraschend, dass die Anfänge der modellgetriebenen Softwareentwicklung mehr als 30 Jahre zurückreichen. Bereits während der 1980er Jahre wurden Computer-Aided Software Engineering (CASE) Werkzeuge genutzt, mit denen einzelne oder mehrere Entwicklungsschritte, wie Analyse und Entwurf (Upper Case Tools) oder Codegenerierung, Testing und Debugging (Lower Case Tools), unterstützt werden konnten (Karagiannis, 2013). Beispielsweise war es möglich, Code Templates aus Modellen zu erstellen, welche dann manuell mit Geschäftslogik befüllt werden mussten. 1989 wurde mit der Gründung der Object Management Group (OMG) ein Konsortium geschaffen, welches mit der Unified Modeling Language (UML), die erste standardisierte grafische Modellierungssprache entwickelt hat.

Daraus sind in weiterer Folge, zwischen 2007 und 2008, die Business Process Model and Notation Specification (BPMN) sowie Domain Specific Languages (DSL) entstanden – diese gelten als wichtige Meilensteine in der modellgetriebenen Softwareentwicklung.

DSLs sind besonders hervorzuheben, da diese im Zusammenspiel mit Codegeneratoren, Interpretern oder Compilern die Modelle in Quellcode umwandeln. DSLs sind dabei, wie der Name bereits sagt, anwendungsspezifische, formale Sprachen – entwickelt um ein bestimmtes Problemfeld (Domäne) bestmöglich darstellen zu können (Fowler & Parsons, Domain-specific languages, 2011). Beispiele für DSLs sind Regular Expressions (RegEx), Structured Query Language (SQL) oder auch Cascading Style Sheets (CSS). Das Gegenteil dieser domänenspezifischen Sprachen sind universell einsetzbare Programmiersprechen, wie Java oder C#.

Abschließen wird in Abbildung 2 noch die Grundidee der modellgetriebenen Softwareentwicklung vorgestellt:

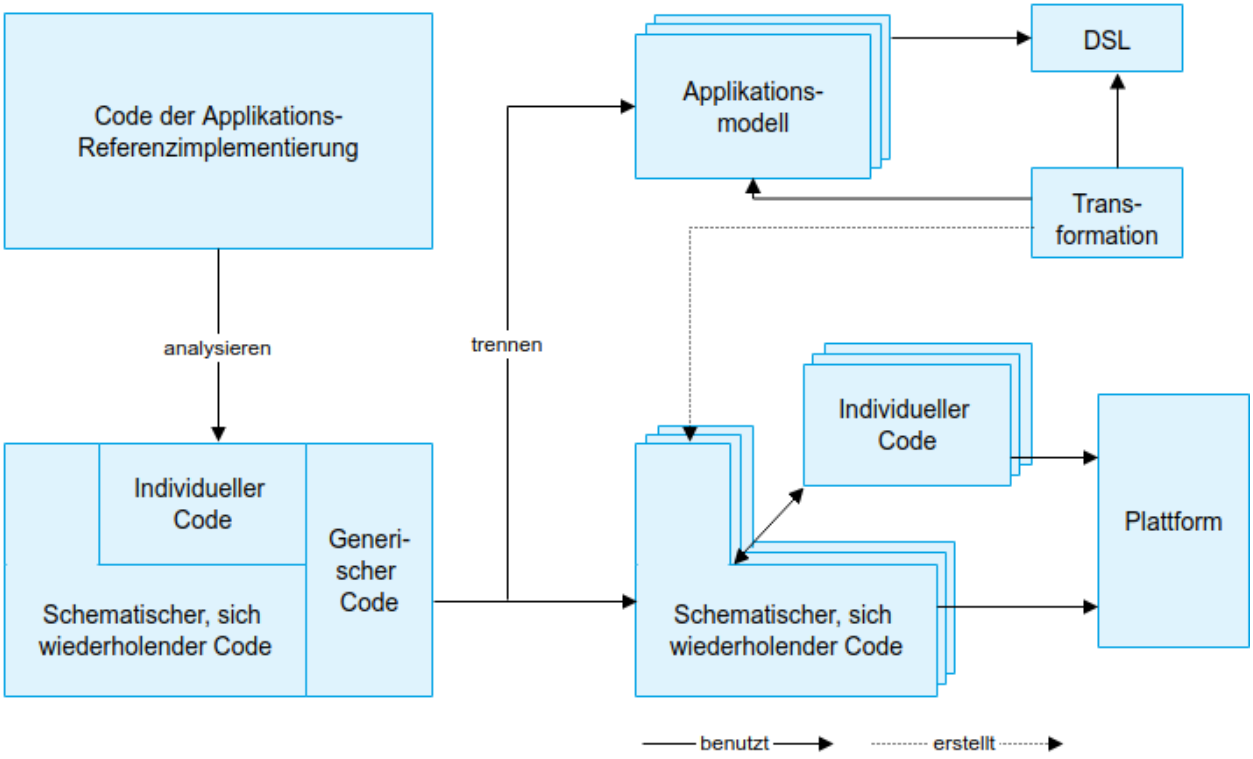

Abbildung 2: Grundidee modellgetriebener Softwareentwicklung (eigene Darstellung in Anlehnung an Stahl, Völter, & Efftinge, 2007)

Betrachtet man existierende Implementierungen bzw. Referenzimplementierungen stellt man fest, dass es sich dabei stets um Unikate handelt, welche eine individuelle Struktur besitzen (Stahl, Völter, & Efftinge, 2007). Der Quellcode dieser Anwendungen kann jedoch beispielsweise durch Refactoring [2] so umstrukturiert werden, dass sich daraus drei Teile ergeben, welche separat betrachtet werden können (in der Grafik unten links angeordnet) (Stahl, Völter, & Efftinge, 2007). Der generische Anteil ist dabei für alle zukünftigen (Teil-) Anwendungen ident. Der schematische Teil ist zwar nicht identisch, jedoch von gleicher Systematik (z.B. basiert auf denselben Entwurfsmustern) und der individuelle Code stellt den anwendungsspezifischen Teil, welcher nicht verallgemeinerbar ist, dar (Stahl, Völter, & Efftinge, 2007). Die MDSD hat dabei das Ziel, den schematischen Anteil für alle (Teil-) Anwendungen jeweils aus einem Anwendungsmodell abzuleiten, wobei es bei der Transformation auch Zwischenstufen geben kann – jedenfalls sind DSL, Transformation und die Plattform die Schlüsselelemente, welche für die jeweilige Domäne einmal erstellt werden müssen (Stahl, Völter, & Efftinge, 2007). Der Code wird also so weit als möglich aus dem Modell generiert. Der restliche Anteil, welcher nicht sinnvoll als Modell abgebildet werden kann, wird manuell kodiert (Dewanto & Klein, Heise Developer, 2018).

2.1.1 **Low-Code / No-Code**

Eine Weiterentwicklung der modellgetriebenen Softwareentwicklung, speziell des Model-centric Ansatzes, sind Low-Code bzw. No-Code Entwicklungsplattformen. Während sich MDSD (Kapitel 2.1) hauptsächlich auf die Produktivität der Entwickler, die Wartbarkeit und Dokumentation der Applikation fokussiert und der einfachere Einstieg in die Materie der Anwendungsentwicklung bei MDSD für Laien nur ein willkommener Nebeneffekt ist, geht man mit Low-Code einen Schritt weiter.

Mit dem Low-Code Ansatz geht man weg vom reinen Entwicklungsprozess, d.h. das Implementieren lauffähiger Software mit einem gewählten Werkzeug anhand zuvor definierter Anforderungen, hin zum gesamten Software-Lebenszyklus (Kapitel 2.4) – diese Plattformen decken also von der Planung über die Entwicklung, der Testphase über das Deployment[3] bis zum Einsatz im laufenden Geschäftsbetrieb alles ab - ganz nach dem Motto „one size fits all“. Man muss also nicht mehrere Werkzeuge nutzen, wie bei der traditionellen Entwicklung üblich, um den gesamten Lebenszyklus abzudecken, sondern hat alle Funktionalitäten innerhalb einer Plattform vereint. Low-Code Entwicklung stellt also eine weitere Möglichkeit dar, um Software schnell und mit minimaler manueller Programmierung zu entwickeln bzw. stellt ein Weg für Entwickler bereit, um rascher und zuverlässiger Ergebnisse zu liefern (Revell, 2019).

[2] Refactoring ist ein Prozess, um ein Softwaresystem so zu verändern, dass das externe Verhalten nicht geändert der Code jedoch eine bessere interne Struktur erhält (Fowler, Refactoring, 2000)

[3] Deployment beschreibt den Prozess der Installation und Konfiguration von Software auf einem Server oder PC

Eine Übersicht der wesentlichen Funktionalitäten, welche eine Low-Code Entwicklungsplattform laut Dewanto & Klein (2018) abdecken sollte, ist in Abbildung 3 ersichtlich. Die Möglichkeit, sämtliche beschriebenen Funktionalitäten als Cloud-Service zu verwenden, ist einer der Vorteile von heutigen Low-Code Entwicklungsplattformen. Eigene Server und Installationen sind zwar möglich, jedoch nicht erforderlich.

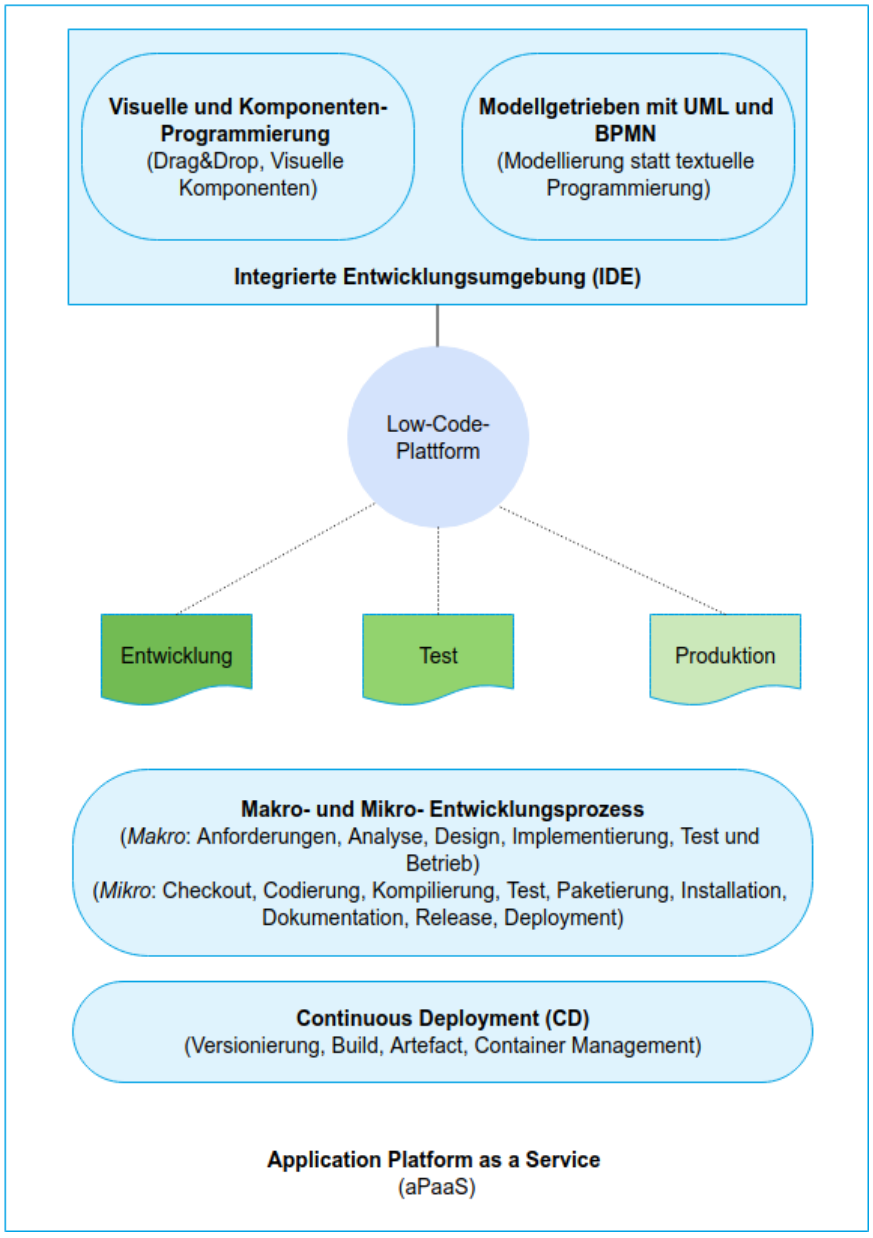

Abbildung 3: Komponenten und Architektur einer Low-Code Plattform (eigene Darstellung in Anlehnung an Dewanto & Klein, 2018)

Eine weitere Verbesserung verglichen mit früheren Vertretern des Model-centric Ansatzes ist es, das Low-Code Plattformen die erstellten Modelle zur Laufzeit interpretieren was bedeutet, dass vorab kein Quellcode mehr generiert werden muss. Der Begriff Low-Code sagt dabei aus, dass, falls nötig, eigener Quellcode in das Modell mit aufgenommen werden kann um spezielle Anforderungen, welche in der Plattform mit den enthaltenen Werkzeugen nicht umsetzbar sind,

abzubilden. Die Versionierung sowie Verpackung des eigenen Codes werden dabei automatisiert von der Plattform erledigt. Im Gegensatz dazu bieten No-Code Entwicklungsplattformen keine solche Möglichkeit der Erweiterung mittels eigenem Quellcode an.

Im Unterschied zur traditionellen Softwareentwicklung, bei der Quellcode individuell geschrieben werden muss, wird bei Low-Code Entwicklungsumgebungen sehr viel Wert auf Wiederverwendbarkeit gelegt. Es stehen vorgefertigte Softwarebausteine zur Verfügung, die frei miteinander kombiniert werden können. Außerdem bieten viele Low-Code Plattformen „Marktplätze“ an, in denen von anderen EntwicklerInnen erstellte Funktionalitäten bezogen werden können.

Eine passende Beschreibung gibt Malcom Ross auf der Website InfoWorld.com, indem er Low-Code Plattformen mit Legobausteinen vergleicht:

> *Think of these pre-built components as a box of Legos. Each block has a specific purpose and can be used in a variety of applications as a piece of a larger puzzle. For example, developers shouldn’t need to code UI objects from scratch when they can simply choose from pre-built UI components. Plus, these pre-built components can be easily reconfigured and updated as needs change* (Ross, 2018).

Neben MDSD können die Wurzeln der Low-Code Entwicklung jedoch auch im Bereich der „fourth-generation-languages“ (4GL) gefunden werden. 4GL bezeichnet dabei Programmiersprachen, welche näher an der menschlichen Sprache sind als andere höhere Programmiersprachen und daher für Personen ohne Ausbildung in der Softwareentwicklung leichter zu erlernen sind (Britannica, 2020). Das gelingt dadurch, dass 4GL Sprachen auf einem höheren Abstraktionslevel als jene der dritten, zweiten und ersten Generation sind und zum Ziel haben, mit möglichst wenig Quellcode für einen bestimmten Anwendungsfall Funktionen oder auch komplette Anwendungen bereitstellen zu können (Industry of Things, 2017). Als erste Generation (1GL) bezeichnet man die Maschinensprache, die zweite Generation (2GL) sind Assemblersprachen bei denen der Quellcode durch so genannte Assembler wieder zurück in ausführbaren Maschinencode (1GL) transformiert wird. Bei der dritten Generation (3GL) handelt es sich um prozedurale Programmiersprachen wie C# oder Java, welche in der Welt der Softwareentwicklung die gängigste Generation darstellt. Im Gegensatz zu MDSD und 4GL sind Low-Code Entwicklungsplattformen jedoch offener und können einfacher in komplexe System-Portfolios integriert werden. Diese Offenheit wird durch bereitgestellte Application Programming Interfaces (APIs) und der Kompatibilität mit open-source Entwicklungsframeworks, wie beispielsweise AngularJS oder React, erreicht.

Weiters muss hier erwähnt werden, dass Rapid Application Development (RAD) Produkte, wie etwa Oracle Forms, bereits in den 1990er Jahren entstanden sind und somit ebenfalls als Vorläufer der Low-Code Entwicklungsplattformen angesehen werden. Bereits mit diesen Programmierumgebungen konnten Entwickler ihre Anwendungen zum Teil visuell erstellen – die Benutzeroberfläche, mit Hilfe eines Designers und unter Zuhilfenahme vorgefertigter Komponenten, welche via Drag & Drop zusammengesetzt wurden. Aber auch die selbst programmierte Geschäftslogik, konnte als eigens entwickelte Komponente in einer Komponentenpalette abgelegt und wiederverwendet werden. Durch die Probleme dieser frühen

RAD-Tools, wie beispielweise deren Proprietarität [4], dem sehr hohen Ressourcenbedarf sowie Geschwindigkeitsproblemen im laufenden Betrieb, haben diese Werkzeuge jedoch relativ schnell wieder an Bedeutung verloren und sind vom Markt verschwunden (Dewanto & Klein, Heise Developer, 2018). Der Begriff RAD wird im IT-Umfeld seit Mitte der 1990er Jahre als Sammelbegriff verwendet, welcher verschiedene Methoden, Techniken und Werkzeuge beschreibt (Berger, Beynon-Davies, & Cleary, 2004). Erstmals wurde RAD jedoch von James Martin (1991) erwähnt, der unter diesem Begriff ein prototypisches Vorgehensmodell entwickelte, welches in Kapitel 2.2.1 genauer beschrieben wird.

In der Theorie bietet die Low-Code Entwicklung einige Vorteile, beispielsweise eine einfachere Wartung des Produktes (Weinmeister, 2015). Der wohl größte Vorteil besteht allerdings darin, dass dem Fachkräftemangel in der IT begegnet werden kann, indem die Hürde, eigene Applikationen zu entwerfen und zu entwickeln, stark gesunken ist.

Neben den bereits oben beschriebenen Aspekten unterscheidet sich Low-Code von MDSD, RAD oder 4GL dadurch, dass im Gegensatz zu den drei letztgenannten eine andere Zielgruppe angesprochen wird (Martin, Application Development Without Programmers, 1982). Diese Zielgruppe wird oft „Citizen Developer“ (CD) genannt und beschreibt AnwenderInnen, die meist direkt in den Fachabteilungen arbeiten und keine professionellen EntwicklerInnen sind. Diese CD erstellen Geschäftsanwendungen mithilfe von Entwicklungswerkzeugen, die zentral von der Unternehmens-IT bereitgestellt werden (Gartner Glossary, 2020).

Wichtig im letzten Satz ist der Punkt, dass die Werkzeuge zentral von der IT bereitgestellt werden – somit soll verhindert werden, dass sich im Bereich der Softwareentwicklung eine so genannte „Schatten-IT“ innerhalb eines Unternehmens etabliert, da es offiziell unterstützte Werkzeuge gibt.

2.1.1.1. Schatten IT

Der Begriff Schatten IT beschreibt dabei jede Form von IT, sei es Hardware oder Software, welche innerhalb eines Unternehmens ohne das Wissen, die Zustimmung oder Unterstützung der IT-Abteilung verwendet wird (Rentrop, Mevius, & Van Laak, 2011). Die Bandbreite reicht hierbei von der Nutzung nicht genehmigter „Sozialer Software“ wie beispielsweise Skype, über die Verwendung von nicht im IT-Katalog enthaltener Hardware bis hin zu eigens entwickelten Anwendungen, welche häufig Access- oder Excel-basiert sind. Grundsätzlich entsteht Schatten-IT, wenn die von der IT-Abteilung angebotenen Services nicht den Anforderungen der Fachabteilungen genügen (Seidel & Reppner , 2009).

Ein solch ungenügender Service kann beispielsweise Zustandekommen, wenn die Abteilung für Softwareentwicklung ein zu volles Backlog[5] aufweist - aufgrund von zu vielen Anforderungen und zu wenig Personal, um diese Anforderungen abzuarbeiten. Eine Fachabteilung, welche Bedarf

[4] Proprietär in Bezug auf Software bedeutet, dass die Zielumgebung (zB ein Betriebssystem) häufig festgelegt ist und das Verlassen dieser Umgebung nur schwer oder eingeschränkt möglich ist (Dewanto & Klein, Heise Developer, 2018).

[5] Unter einem Backlog versteht man Ansammlung von Aufgaben bzw. Anforderungen, welche abgearbeitet werden müssen.

an einer neuen Lösung angemeldet hat, hat somit zwei Vorgehensmöglichkeiten: 1) Warten bis die Abteilung für Softwareentwicklung die Anforderung einplant und umsetzt oder 2) die eigene Anforderung selbständig, mit den zur Verfügung stehenden Mitteln, abarbeiten. Da diese zur Verfügung stehenden Mittel oft begrenzt sind (Excel, Access, …), sind damit auch Einschränkungen bezüglich der Funktionalität und Effektivität verbunden. Die Integration solcher Insellösungen in ein größeres System ist schwierig, ebenso erschwert es die abteilungsübergreifende Zusammenarbeit sowie Wartbarkeit der Anwendung. Im schlimmsten Fall gibt es keinen „Single Source of Truth“ (SSOT), das bedeutet keinen allgemeingültigen Datenstand.

Diesem Problem, und zwar der Entwicklung von eigenen Lösungen in der Fachabteilung ohne Wissen und der Unterstützung der IT, kann mit Low-Code Plattformen begegnet werden. Durch das von der IT-Abteilung zur Verfügung gestellte Werkzeug, können die negativen Effekte der Schatten-IT, wie beispielsweise Probleme mit der Datensicherheit, SSOT oder die daraus resultierenden Compliance Konflikte vermieden werden, ohne das die Chancen, wie die hohe Innovationsrate, die starke Fokussierung auf Prozesse und die daraus resultierende Benutzerzufriedenheit, verloren gehen (Rentrop, Mevius, & Van Laak, 2011). Ein weiterer Vorteil, welcher durch das Einbeziehen der IT-Abteilung und das Nutzen der bereitgestellten Werkzeuge entstehen kann ist, das beispielsweise eine bereits bestehende Lösung aus Abteilung A, die ebenso für Abteilung B interessant wäre, sichtbar gemacht werden kann. Somit können Doppelentwicklungen und der damit verbundene Kosten- und Zeitverlust vermieden werden.

2.1.2 **Mendix**

Als Ergebnis einer Evaluierung, welche innerhalb der ANDRITZ AG verschiedene Low-Code Entwicklungsplattformen miteinander verglichen hat, wird seit Anfang 2018 Mendix (Mendix, 2020) als Lösung eingesetzt. Low-Code Anbieter werden jährlich von IT-Beratungsunternehmen wie Gartner (Gartner, 2019) anhand ihrer Unternehmensvision und deren Umsetzung bewertet. Das Ergebnis ist der Gartner Magic Quadrant, eine Einteilung in vier Abschnitte (Leaders, Challengers, Niche-Players und Visionaries), welche widerspiegelt wie ausgereift bzw. innovativ eine Low-Code Plattform ist. Eine weitere Analyse liefert das Marktforschungsunternehmen Forrester (Forrester, 2019) mit der Forrester Wave, welche unter anderem auch Befragungen von Referenzkunden oder die Marktstellung in das Ergebnis mit einfließen lässt. Der Gartner Magic Quadrant aus dem Jahr 2019 ist in Abbildung 4 ersichtlich:

Abbildung 4: Gartner „Magic Quadrant for Enterprise Low-Code Application Platforms" (Vicent, Kimihiko, Wong, Yefim , & Driver, 2019)

Neben der Tatsache, dass Mendix in beiden der o.a. Analysen im Leader Bereich angesiedelt ist und dementsprechend eine der ausgereiftesten Plattformen am Markt bietet, hat, neben finanziellen und weiteren Aspekten, noch ein Faktor zur Entscheidung der ANDRITZ AG Mendix als Werkzeug zu verwenden geführt: Die strategische Partnerschaft von Mendix mit SAP und der daraus resultierenden Möglichkeit, ohne Funktionalitäten von Drittanbietern, mit Daten der ANDRITZ SAP Umgebung zu interagieren.

Mit Mendix ist es möglich, webbasierte Anwendungen wie Responsive Websites oder mobile Applikationen zu erstellen. Die Modelle, welche mit Mendix erstellt werden, basieren auf der objektorientieren Programmiersprache Java - Mendix bietet somit alle Vorteile der OOP. Es kann dabei jederzeit durch den von Mendix zur Verfügung gestellten Software Development Kit (SDK), auf die erzeugten Modellen und die verwendeten Daten zugegriffen werden – somit ist gewährleistet, dass falls Mendix als Plattform innerhalb eines Unternehmens nicht weiter verwendet wird, die bereits erstellten Anwendungen hin zu einer anderen Low-Code Plattform oder direkt in eine 3GL Sprache migriert werden können. Durch diese SDK und verschiedenen Schnittstellen (APIs), mit denen beispielsweise Continuous Integration (CI) oder Continuous Delivery (CD) Tools – siehe Kapitel 2.4 - von Drittanbietern angebunden oder erstellte Modelle manipuliert werden können, ist Mendix als Plattform sehr offen und erweiterbar, was in Abbildung 5 dargestellt wird.

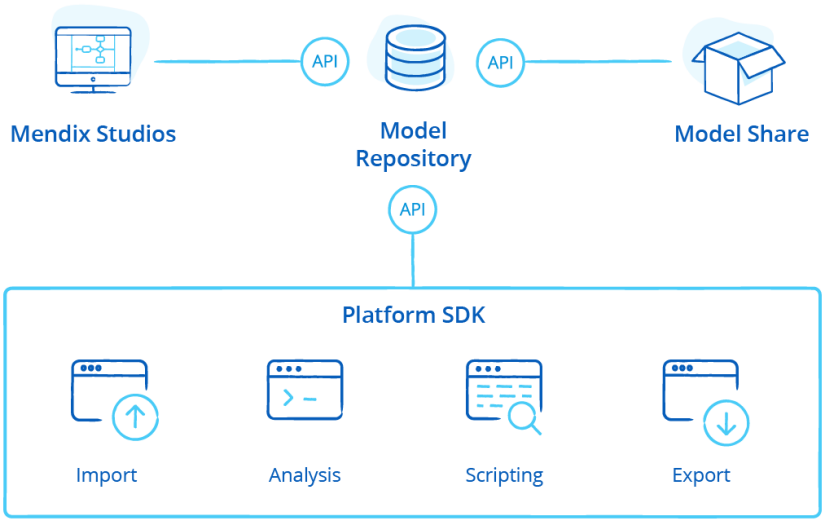

Abbildung 5: Offenheit und Erweiterbarkeit der Mendix Plattform (Medix Evaluation Guide, 2019)

Folgende Elemente innerhalb der Mendix Plattform werden eingesetzt, um eine Anwendung zu entwickeln (Mendix Docs, 2020) :

- Domain Model: Nutzt eine Mendix spezifische Notation, welche an UML Diagramme angelehnt ist, um die Datenstruktur der Anwendung abzubilden. Das Domain Model besteht aus Entitäten, welche Attribute aus verschiedenen Datentypen enthalten können. Die Entitäten spiegeln dabei eine Tabelle in einer relationalen Datenbank wider, die Attribute sind die Spalten in dieser Datenbank. Zusammenhänge zwischen den einzelnen Entitäten werden mittels Assoziationen wiedergespiegelt, welche den Relationen in der Datenbank entsprechen.
- Microflows / Nanoflows: Mit dieser an die BPML Notation angelehnte visuelle Modellierungsweise, wird die Geschäftslogik innerhalb der Anwendung implementiert. Microflows führen die Logik dabei am Server aus, Nanoflows direkt am Client. Es steht ein Set aus vorgefertigten Aktionen zur Verfügung, welche alle Funktionalitäten, die aus der objektorientieren Programmierung bekannt sind, abdeckt (Objekte erstellen, Iterieren, Datenbankabfragen, logische Entscheidungen, …).
- Pages: Stellen das User-Interface dar, welches mit einem WYSIWYG Editor erstellt werden kann. Es werden spezielle Views (z.B. Data-View für ein einzelnes Objekt, List-View für mehrere Objekte) verwendet, um die Daten, welche entweder direkt aus der Datenbank oder via Microflow bereitgestellt werden, im Frontend darzustellen. Darüber hinaus stehen wiederverwendbare Blöcke zur Verfügung, welche nahezu alle aus HTML5 bekannten Gestaltungsmöglichkeiten bieten (z.B. Div-Container, Tabellen, Input Felder, Buttons, Video Player usw.). Außerdem ist es möglich via Cascading Stylesheets (CSS) das Aussehen der Applikation jederzeit an z.B. die Corporate Identity anzupassen.
- Ressources: Beinhalten Funktionen bzw. Helfer, welche in z.B. Microflows/Nanoflows oder im Domain Model wiederverwendet werden können. Darunter fallen beispielsweise

Regular Expressions (RegEx), eigens geschriebene Java Aktivitäten, Aufzählungen, welche als „Enumeration“ Datentyp verwendet werden können usw.

Die Zusammenarbeit zwischen mehreren Entwicklern wird über die Plattform eigene Versionierung, basierend auf Subversion (SVN), gehandhabt. Das zentrale Projektarchiv (Repository) ist dabei in der Mendix Cloud gehostet und kann über den so genannten Team Server verwaltet werden.

Wie bereits im Kapitel 2.1 (Low-Code / No-Code) kurz erwähnt, bietet Mendix auch die Möglichkeit Funktionalitäten, welche von anderen Nutzern entwickelt wurden, aus einem in der Plattform integrierten App Store zu beziehen. Diese Funktionalitäten werden Widgets genannt. Das Spektrum reicht dabei von kleinen Anzeigehelfern, wie einem animierten Fortschrittsbalken oder einem Bilderkarussell, bis hin zu Konnektoren mit denen Machine Learing (z.B. Azure Face API) oder Internet-of-Things (z.B. AWS IoT Connector) Anwendungen erstellt werden können. Diese Funktionalität ist im Mendix Studio Pro nutzbar, welches als Low-Code Entwicklungsplattform angesehen und als Desktop Anwendung installiert werden muss – die Zielgruppe dieser Umgebung sind erfahrene Entwickler, welche meist direkt in der IT-Abteilung angesiedelt sind. Für die Entwickler der Fachabteilung ohne Entwicklungserfahrung, steht mit Mendix Studio eine No-Code Umgebung zur Verfügung, welche online nutzbar ist und eine einfachere Bedienung mit eingeschränkten Möglichkeiten bietet.

Das Deployment der Anwendung auf die gewünschte Cloud Umgebung, das bedeutet das Erstellen des Paketes sowie das Verschieben des Paketes auf die gewünschte Umgebung, wird ebenfalls von der Plattform mittels eigenem Build Server erledigt. Die erstellten Pakete werden dabei in einem eigenen Repository verwaltet.

Des Weiteren beinhaltet Mendix Funktionalitäten, welche die Zusammenarbeit der NutzerInnen erleichtern soll – so ist es beispielsweise möglich, User Stories direkt in der Mendix Plattform zu erstellen und einem / einer EntwicklerIn zuzuweisen.

Eine Übersicht der oben beschriebenen Funktionalitäten ist in Abbildung 6 ersichtlich:

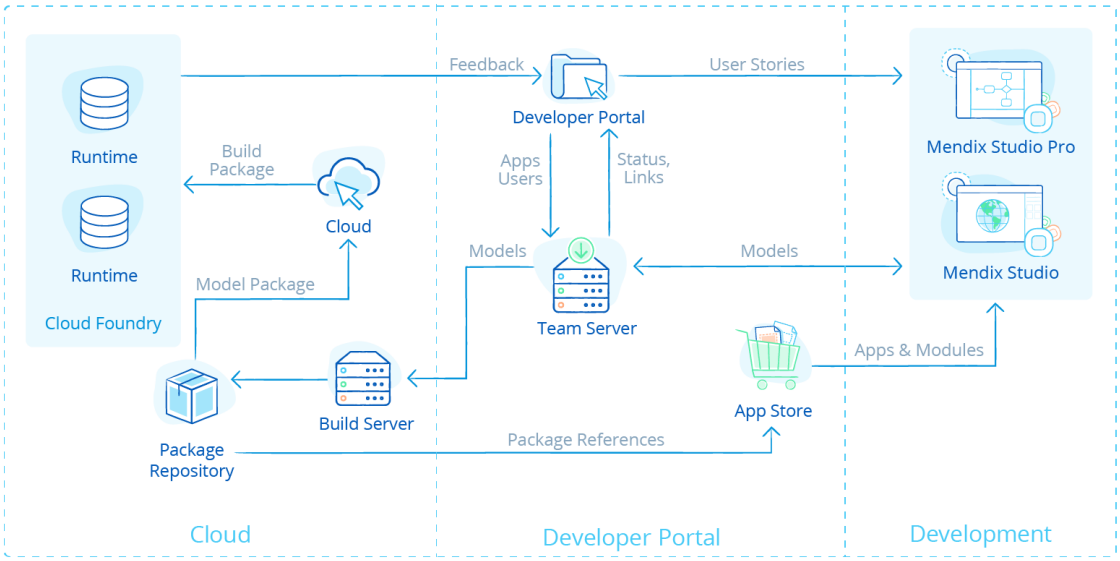

Abbildung 6: Module & Services der Mendix Plattform (Medix Evaluation Guide, 2019)

2.2 Vorgehensmodelle in der Softwareentwicklung

Unter einem Vorgehensmodell in der Softwareentwicklung versteht man einen standardisierten Ablauf, welchen ein Unternehmen verfolgt, um ein Vorhaben umzusetzen - diese Abläufe stellen unabhängig vom Vorhaben sicher, dass die Aufgaben ohne Überspringen von Teilschritten durchgeführt und abgeschlossen werden (Wieczorrek & Mertens, 2011). In der Informationstechnik (IT) generell und speziell in der Softwareentwicklung, wurden im Laufe der Jahre bereits etliche solcher standardisierten Verläufe entwickelt und dokumentiert. Einige dieser Vorgehensmodelle sind sehr wissenschaftlich-theoretisch, andere bereits praxisnah erprobt (Berg, Knott, & Sandhaus, 2014). Solch ein Vorgehensmodell begleitet ein Softwareprodukt optimalerweise durch den gesamten Software-Lebenszyklus, welcher in Kapitel 2.4 genauer betrachtet wird. Grob können die vorhandenen Vorgehensmodelle zur Softwareentwicklung in die Kategorien traditionelles- und agiles Projektmanagement eingeteilt werden.

Bei traditionellen bzw. klassischen Vorgehensmodellen, wie beispielsweise dem Wasserfallmodell, den V-Modell oder dem Spiralmodell, werden die einzelnen Phasen eines Projektes sequenziell abgearbeitet. Die Planung des Projektes findet zur Gänze zu Projektbeginn statt. In der Literatur werden diesen traditionellen Vorgehensmodellen einige Vorteile zugeschrieben, beispielsweise werden damit langfristige Planungen für Projekte, welche stabile Rahmenbedingungen benötigen, möglich (Boehm & Turner, 2004). Außerdem sind sie sind gut einsetzbar für kritische, risikoreiche und große Projekte (Berg, Knott, & Sandhaus, 2014). Nachteile dieser Methoden sind beispielsweise der erhöhte organisatorische und planerische Aufwand (Berg, Knott, & Sandhaus, 2014) sowie das diese Methoden nur schwer auf kleine Projekte skalierbar sind (Berg, Knott, & Sandhaus, 2014) oder das ein Rückschritt auf eine bereits abgeschlossene Phase nicht vorgesehen bzw. aufwändig ist (Tiemeyer, 2014).

Bei Vertretern agiler Methoden, wie beispielsweise Scrum oder Kanban, wird stattdessen iterativ vorgegangen und die Projektplanung findet nicht komplett zu Projektbeginn statt. Diese Art des

Projektmanagements bietet eine hohe Flexibilität, wodurch rasch auf Änderungswünsche seitens des Kunden reagiert werden kann (Tiemeyer, 2014) und ist gut geeignet für Projekte, bei denen die Anforderungen noch nicht klar spezifiziert sind sowie für Projekte, die aus einem eher kleinen Team bestehen (Boehm & Turner, 2004). Durch die vermehrte und aktive Kommunikation mit dem Kunden entsteht vertrauen (Tiemeyer, 2014). Natürlich haben diese agilen Vorgehensmodelle auch Nachteile, zum Beispiel sind sie auf große Teams und Projekte schwer skalierbar (Berg, Knott, & Sandhaus, 2014), außerdem ist eine langfristige Planung sowie das definieren konkreter Termine schwierig (Eckstein, 2015).

Welche Anforderungen ein Vorgehensmodell generell erfüllten sollte, um als solches Wahrgenommen zu werden, ist in Abbildung 7 ersichtlich:

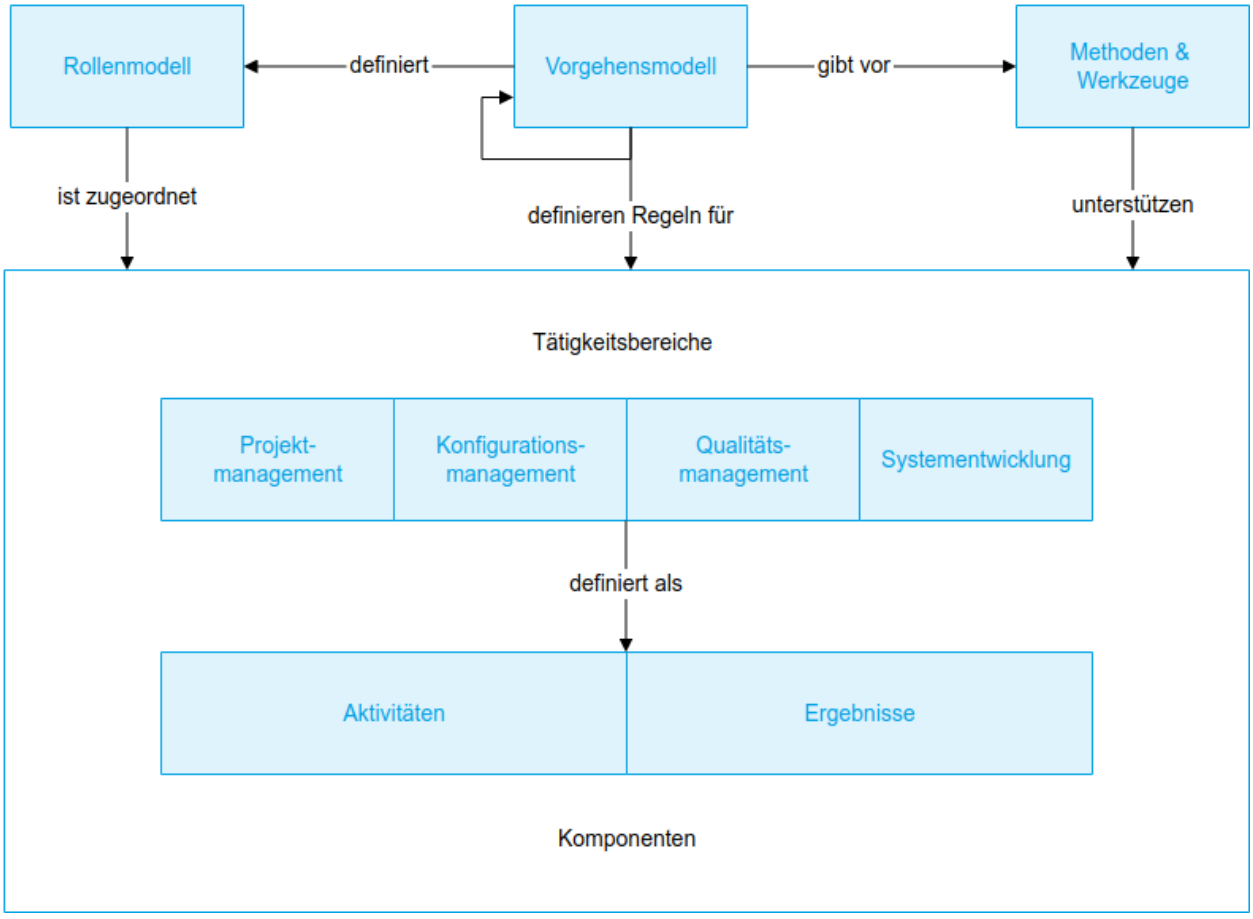

Abbildung 7: Anforderungen an ein Vorgehensmodell (eigene Darstellung in Anlehnung an Balzert, 2009)

Die einzelnen Tätigkeitsbereiche, welche in der o.a. Abbildung ersichtlich sind, können sich je nach Domäne bzw. Branche natürlich deutlich voneinander unterscheiden. Um ein solches Vorgehensmodell erfolgreich anwenden bzw. umsetzen zu können, ist es notwendig, dass ein potenzieller Anwender die definierten Rollen, Methoden & Werkezuge sowie die damit verbundenen Tätigkeitsbereiche und Komponenten als Ganzes versteht. Wie bereits in Kapitel 2.1 modellgetriebene Softwareentwicklung erwähnt, helfen Modelle die Komplexität der Realität zu reduzieren. Dies gilt auch für Vorgehensmodelle, damit soll es ermöglicht werden einen

konzentrierten Blick auf wichtige Prozesse und Methoden zu werfen (Berg, Knott, & Sandhaus, 2014).

In vielen Projekten, in denen Software entwickelt wird, werden heutzutage agile Methoden eingesetzt – neben den bereits zu Beginn dieses Kapitels genannten Vorteilen, sprechen noch folgende Gründe für deren Einsatz (Humble & Farley, 2011):

- Durch die Priorisierung der Anforderungen nach wirtschaftlichen Gesichtspunkten wird die zu entwickelnde Software oft schon vor Projektende als nützlich empfunden.
- Durch das regelmäßige Feedback der Stakeholder finden laufend Anpassungen der Anforderungen statt. Das ist auch nötig, da zu Projektstart die Anforderungen noch recht unklar bzw. unscharf sind, auch für die Auftraggeber. Es entwickelt sich erst während des Projektes ein klares Bild darüber, was benötigt wird. Diese Entwicklung in Abbildung 8 ersichtlich ist:

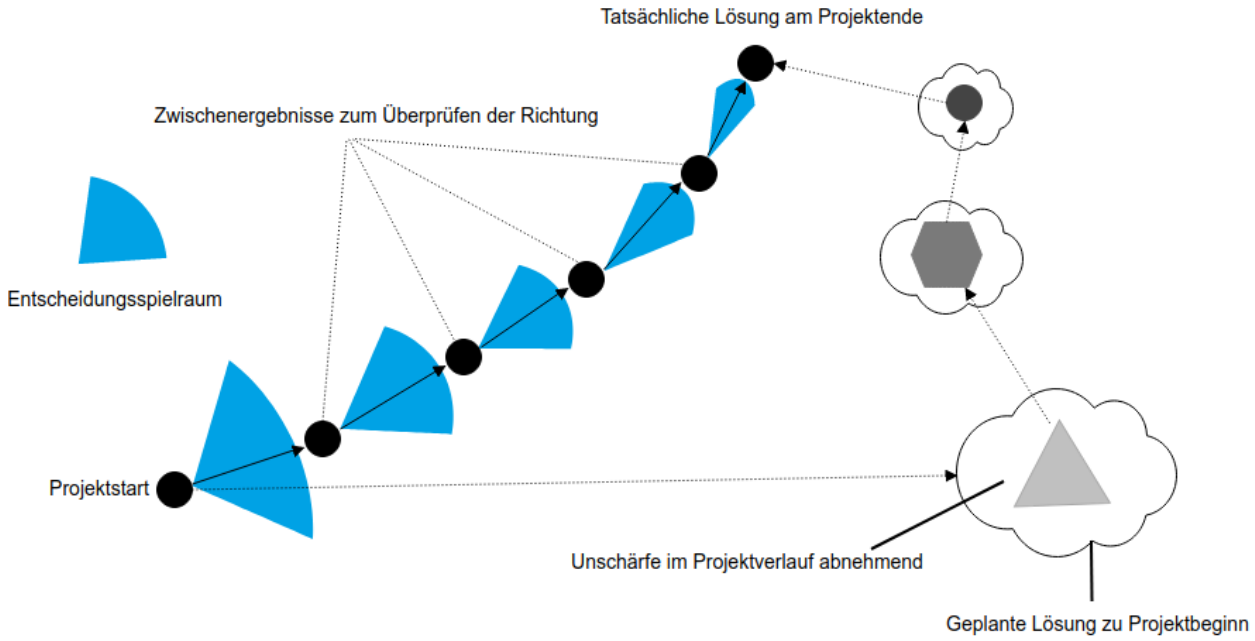

Abbildung 8: Entwicklung von Softwareprojekten (eigene Darstellung in Anlehnung an Starke, 2018)

- Aufgaben gelten in der agilen Entwicklung erst dann als erledigt, wenn diese auch vom Kunden abgenommen sind – somit ist die Gefahr, dass Fehlentwicklungen passieren, geringer.

All die in diesem Kapitel erwähnten Vorgehensmodelle sind in der Softwareentwicklung weit verbreitet, im Fall der Low-Code bzw. No-Code Entwicklung hat sich jedoch RAD als weit verbreitetes Vorgehensmodell etabliert. Die Gründe dafür werden im folgenden Kapitel näher erläutert.

2.2.1 Rapid Application Development

Um die Vorteile einer Low-Code Plattform optimal nutzen zu können, ist es notwendig, ein Vorgehensmodell zu wählen, welches die Schnelligkeit und Flexibilität der Entwicklungsumgebung optimal unterstützt. Als RAD 1991 erstmals von James Martin in seinem gleichnamigen Buch „Rapid Application Development“ erwähnt wurde war schnell klar, dass damit viel schneller qualitativ hochwertigere Resultate erzielt werden konnten als mit den zu dieser Zeit üblichen Vorgehensmodellen, wie z.B. dem Wasserfallmodell (Martin, Rapid Application Development, 1991).

Die Grundidee hinter RAD ist ein prototypisches Vorgehensmodell und basiert auf dem Spiralmodell, welches in den 1980er Jahren von Barry Boehm entwickelt wurde und einen agilen und iterativen Softwareentwicklungsprozess beschreibt (Ruparelia, 2010). Die beiden Hauptgründe, welche zur Entwicklung des RAD Modells führten, waren einerseits die Betriebsunsicherheit, andererseits die Entwicklungsunsicherheit (Beynon-Davies, Mackay, & Tudhope, 2000). Unter Betriebsunsicherheit versteht man die schnelllebige Umwelt in der Unternehmen rasch auf Veränderungen reagieren müssen und Entwicklungsunsicherheit beschreibt die sich ständig erweiternde Anzahl an Technologien und Tools, welche zum Entwickeln von Anwendungen genutzt werden können (Beynon-Davies, Mackay, & Tudhope, 2000). Beiden Unsicherheiten konnte mit dem Wasserfallmodell, bei dem die Entwicklung durch den linearen Ansatz und die klare Abgrenzung der einzelnen Phasen oft sehr lange Zeit in Anspruch genommen hat, nicht begegnet werden. Einer der Vorteile von RAD ist, dass im Vergleich zu anderen Vorgehensmodellen, durch den prototypischen Ansatz weniger an Vorarbeit benötigt wird und dadurch schneller produktive Ergebnisse zu erwarten sind.

Die Phasen des RAD Modells laut Martin sind in Abbildung 9 ersichtlich und teilen sich wie folgt auf:

- Requirements Planning Phase: Obwohl die Phase kürzer als in anderen Vorgehensmodellen ausfällt, ist sie trotzdem von entscheidender Bedeutung für den Projekterfolg. Alle am Projekt beteiligten Personen (Entwickler, Projektleiter, Kunden) sind gefordert die Projektziele und den Projektscope[6] festzulegen, sowie die Anforderungen an das Endprodukt zu definieren. Außerdem ist es wichtig, bereits in dieser Phase auf mögliche Probleme, welche sich aus den Anforderungen ergeben können, hinzuweisen. Diese Phase endet, sobald die Hersteller der Software sowie die Kunden sich auf die Ziele und Anforderungen geeinigt haben.
- User Design Phase: Während dieser Phase arbeiten die Kunden sehr eng mit den EntwicklerInnen zusammen, um Prototypen für alle wichtigen Geschäftsprozesse und die User Ein- sowie Ausgabemasken zu erstellen. Diese werden parallel, je nach Anzahl der verfügbaren Entwickler, erstellt. Meist sind mehrere Iterationen notwendig, um alle Fehler

[6] Unter Projektscope versteht man Inhalts- und Umfangsmanagement – es soll sicherstellen, dass genau die Aufgaben durchgeführt werden, welche für einen erfolgreichen Projektabschluss notwendig sind.

in den Prototypen auszumerzen und den Kunden das gewünschte Ergebnis zu liefern. Durch den intensiven Kontakt in dieser Phase lernen die Entwickler sehr genau, was die Kunden benötigen. Anders herum sehen die Kunden jedoch auch klar, was machbar ist und was nicht.

- Construction Phase: Hier wird aufbauend auf den Prototypen aus der User Design Phase die eigentliche Software entwickelt. Da die meisten Probleme bzw. Missverständnisse bereits in Phase 2 durch die Iterationen der Prototypen aufgelöst wurden, kann das eigentliche Produkt schneller entwickelt werden (Martin, Rapid Application Development, 1991). In dieser Phase ist es weiterhin möglich, Änderungen seitens des Kunden in das Projekt einfließen zu lassen.
- Cutover Phase: Stellt die letzte Phase im RAD Modell dar und beinhaltet die nötigen Schritte, um das Produkt für den Kunden nutzbar zu machen. Das bedeutet, Deployment auf das Zielsystem, Einspielen von Daten, Testen auf dem Zielsystem sowie dem abschließenden Training der Nutzer.

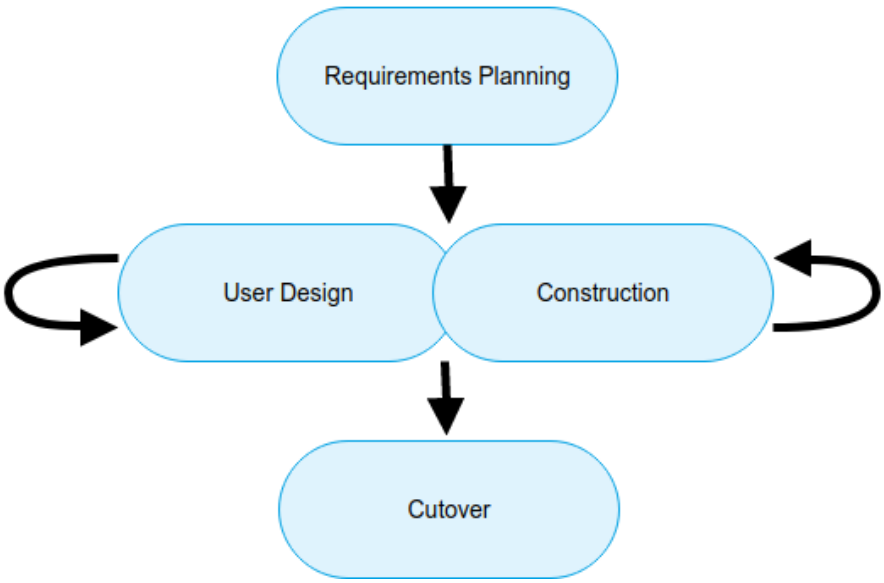

Abbildung 9: RAD Phasen (eigene Darstellung in Anlehnung an Martin, 1991)

Speziell durch den prototypischen bzw. „try before you buy“ Ansatz eignet sich RAD hervorragend für den Einsatz mit Low-Code Plattformen. Durch die vorgefertigten Bausteine können KundInnen sehr schnell Ergebnisse präsentiert werden, welche bei Bedarf ohne viel Aufwand wieder geändert werden können. Es wird, im Gegensatz zu beispielsweise SCRUM oder Kanban, weniger Zeit in die genaue Spezifikation der Anforderungen investiert. Stattdessen werden den KundInnen laufend Prototypen präsentiert, welche anschließend iterativ verfeinert werden. Wenn es mit Low-Code Plattformen möglich ist, Anwendungen in Echtzeit vor den Augen der Anwender oder gar mit diesen gemeinsam live am Bildschirm umzubauen, dann ist der Scrum typische Rhythmus von 2 Wochen (Sprints) zu langsam (Noack, 2019). Ebenso ist eine direkte Interaktion zwischen AnwenderInnen und EntwicklerInnen, beispielsweise um gemeinsam optimale

Lösungswege zu finden - das bedeutet für den / die AnwenderIn sinnvolle und zugleich für den / die EntwicklerIn technisch umsetzbare Lösungen - in Scrum nicht vorgesehen (Noack, 2019). Jedoch hat auch dieses Vorgehensmodell Nachteile, abhängig von der Anzahl der nötigen Iterationen der Prototypen, kann es passieren, dass die geschätzten Zeiten bzw. das zur Verfügung gestellte Budget überschritten wird.

2.3 Software-Lebenszyklus

Wie im vorigen Kapitel kurz angedeutet, existieren viele verschiedene Modelle zur Softwareentwicklung – jedes davon mit Vor- und Nachteilen, abhängig davon welche Software wie schnell, mit welchem Team und welchem Budget entwickelt werden soll. Eines jedoch haben alle diese Modelle gemeinsam, sie orientieren sich an den einzelnen Phasen des Software-Lebenszyklus.

Jedes Softwareprodukt durchläuft während seiner Entstehung verschiedene Teilschritte (siehe Abbildung 10), welche für nahezu alle anderen Softwareprojekte ebenfalls notwendig sind, um ein erfolgreiches Produkt zu erstellen. Aus diesen einzelnen Teilschritten kann ein allgemeiner Software-Lebenszyklus abgeleitet werden.

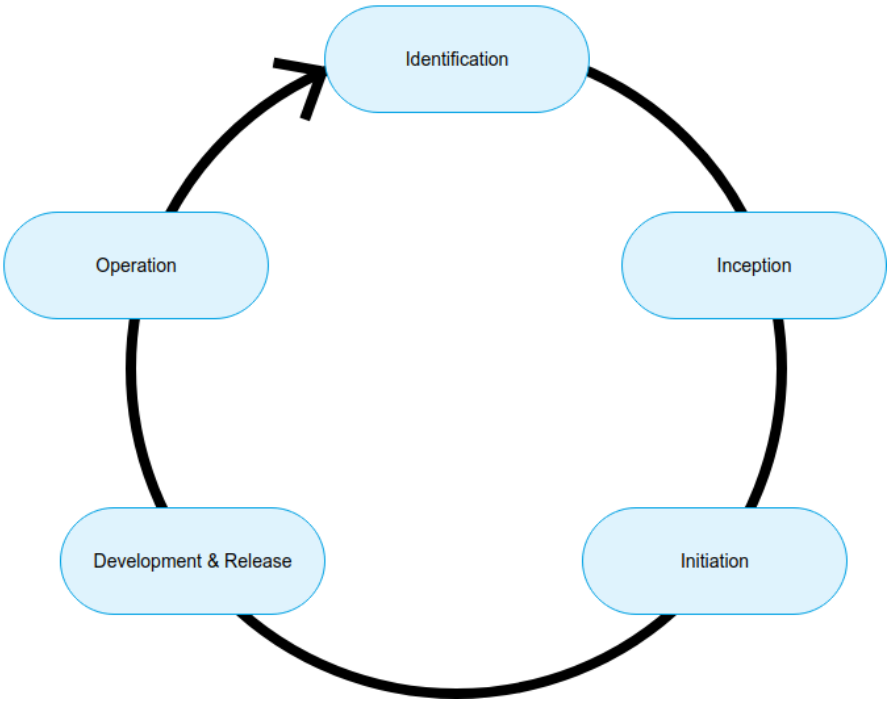

Abbildung 10: Software-Lebenszyklus (eigene Darstellung angelehnt Humble & Farley, 2011)

Der Lebenszyklus beginnt im Regelfall mit der Spezifikation des Produktes und endet nach der Wartungsphase, das bedeutet, das Produkt wird abgeschafft, abgelöst oder weiterentwickelt (Balzert, 2011). Die einzelnen Phasen werden in den folgenden Kapiteln kurz beschrieben.

2.3.1 Identification

In dieser Phase soll ein gemeinsames Verständnis über das zu entwickelnde System geschaffen werden, um in der darauffolgenden Phase (Inception) das Interesse aller relevanten Stakeholder angemessen repräsentieren zu können (Humble & Farley, 2011). Es ist ratsam alle Personen, die vom Projekt betroffen sind, aufzulisten, da wichtige Stakeholder, welche in dieser Phase vergessen werden, in den folgenden Phasen das Projekt durch nicht wahrgenommene Anforderungen stark verzögern und im Extremfall sogar zum Abbruch bewegen können. Fragen, welche in dieser Phase geklärt werden müssen, sind beispielsweise

- Wer sind die zukünftigen Benutzer des Systems?
- In welcher Art und Weise wird das System genutzt werden?
- Was sind die relevanten Daten für Import bzw. Export?

2.3.2 Inception

Aufbauend auf die vorherige Phase werden in der Inception Phase die konkreten Anforderungen an das zu entwickelnde System gesammelt, analysiert, dokumentiert und geplant. Als Resultat dieser Phase sollen Artefakte in einem Detaillierungsgrad entstehen, die es ermöglichen, mit der eigentlichen Arbeit am Projekt, d.h. dem Entwickeln des Systems, zu beginnen. Solche Artefakte sind lebende Dokumente, welche sich im Laufe der Entwicklung ständig ändern und müssen daher stets aktuell gehalten werden (Humble & Farley, 2011). Beispiele für solche Artefakte sind etwa ein Release Plan, eine Test-Strategie, nicht funktionale Anforderungen (z.B. Sicherheit, Verfügbarkeit, …) an das System sowie ein Business Case, welcher den erwarteten Mehrwert des Projektes wiedergibt. Die erstellten Artefakte werden anschließend mit den relevanten Stakeholdern (KundInnen, EntwicklerInnen, Management, Operations, Wartungsteam, …) abgestimmt, um ein gemeinsames Projektverständnis zu gewährleisten.

2.3.3 Initiation

In dieser Phase wird die benötigte Projektinfrastruktur geschaffen und beinhaltet auszugsweise folgende Tätigkeiten (Humble & Farley, 2011):

- Bereitstellung der nötigen Hardware und Software sowie der Basis Infrastruktur, wie beispielsweise Zugang zu benötigten Systemen (Entwicklungs-, Qualitäts- und in späterer Folge das Livesystem) für alle relevanten Teammitglieder.
- Rollen und Verantwortlichkeiten, Arbeitszeiten und Meeting Strukturen müssen definiert werden.
- Erstellen des Anforderungsbacklogs und ein erneuter Blick auf das zu entwerfende System, um dessen Potentiale zu erkennen.
- Identifikation der Risiken (Analyse-, Entwicklungs- und Testrisiken) sowie, soweit möglich, Minderung dieser

2.3.4 **Development & Release**

Mit der Schaffung der Voraussetzung in den vorigen Phasen, kann in dieser Phase mit der eigentlichen Entwicklung begonnen werden. Abhängig vom gewählten Vorgehensmodell, beispielsweise RAD, Scrum oder dem Wasserfallmodell, kann sich das Vorgehen in dieser Phase stark voneinander unterscheiden. Auf eine detaillierte Ausführung dieser Phase wird daher verzichtet.

2.3.5 **Operation**

In den allermeisten Fällen ist das erste Release einer Software nicht das letzte. Daher ist es stark vom Projekt abhängig, was in dieser Phase passiert. Es kann durchaus sein, dass sich die operative Phase nicht von der eigentlichen Entwicklung unterscheidet – es werden im Normallfall weiterhin Features entwickelt und weitere Releases gebaut (Humble & Farley, 2011).

Ein wichtiger Bestandteil dieser Phase ist jedoch das Monitoring der Software. Es werden Parameter definiert und überwacht, welche notwendig sind, um die Performance und Verfügbarkeit des Systems zu beurteilen. Darauf aufbauend können durch Warnsysteme frühzeitig Maßnahmen ergriffen werden, sollten die Antwortzeiten des Systems zu hoch sein oder die generelle Erreichbarkeit des Systems gefährdet sein.

Ein häufiges Problem in dieser Phase stellt die Abhängigkeit des Entwicklungsteams zu einem anderen Team, welches den Betrieb der Anwendungen verantwortet, dar. Die vielfach eingesetzten agilen Entwicklungsmodelle bringen nur dann einen Mehrwert, wenn die Ergebnisse auch schnell in der Produktion eingesetzt werden können (von Kielpinski, 2018). Um das zu erreichen, muss innerhalb eines Unternehmens ein kollaborativer Ansatz zwischen Entwicklung und Betrieb etabliert werden, welcher das in manchen Unternehmen typische „Silodenken“ [7] durchbricht. Unter anderem aus dieser Notwendigkeit ist „DevOps“ entstanden, welches im folgenden Kapitel kurz beschrieben wird.

[7] Silodenken beschreibt ein auf die eigene Abteilung und Tätigkeit fokussiertes Denken und Handeln welches die Zusammenarbeit zwischen einzelnen Unternehmensbereichen erschwert

2.4 DevOps

DevOps ist ein aus den beiden Begriffen Development (Entwicklung) und Operations (Betrieb) zusammengesetztes Kunstwort und bedeutet vor allem, Kommunikation und Kollaboration über Abteilungsgrenzen und Verantwortlichkeiten hinweg - es geht darum, Feedback, Offenheit, Transparenz und eine offene Fehlerkultur direkt in die Entwicklung einfließen zu lassen (von Kielpinski, 2018). DevOps ist mehr als Kultur bzw. als Wertesystem zu betrachten, welches durch Methoden und Prozesse unterstützt wird und weniger als eine eindeutige Methode oder Prozess. Das wird auch deutlich, wenn man sich das CALMS Modell ansieht, welches einen konzeptionellen Rahmen für die Etablierung von DevOps innerhalb eines Unternehmens definiert: (van Herpen, den Broeder , & van Ewijk , 2016)

- **C**ulture (Kultur) – beschreibt die unternehmensweit geltenden Werte, Überzeugungen und Handlungen. Wie gehen Personen innerhalb eines Teams bzw. einer Organisation miteinander um und welche gemeinsamen Werte teilen sie? Hier sollen „Verhaltensregeln“ definiert werden, um das zuvor erwähnte Silodenken zu vermeiden.
- **A**utomation (Automatisierung) – MitarbeiterInnen sollen, durch das Automatisieren von sich wiederholenden Aufgaben, von diesen befreit werden. Ziel ist es, Fehlerquellen zu reduzieren und Prozessabläufe zu beschleunigen.
- **L**ean („Schlank“) – Vermeidung von Verschwendung. Das bedeutet, nicht wertbeitragende Aktivitäten so gut als möglich zu vermeiden bzw., wenn vermeiden nicht möglich ist, diese zumindest zeitlich zu optimieren. Das passiert einerseits durch den zuvor erwähnten Punkt der Automatisierung, andererseits ist es auch möglich beispielsweise die Häufigkeit und Länge von Besprechungen zu optimieren oder die Anzahl der verwendeten Werkzeuge auf ein Minimum zu reduzieren.
- **M**etrics (Metriken) - Hier geht es um das Sammeln und Analysieren von quantitativen Daten, um Rückschlüsse auf die Effizienz der eingesetzten Prozesse und Methoden ziehen zu können. Ziel ist es, sich laufend nachweislich zu verbessern.
- **S**haring (Teilen) – Durch die unterschiedlichen Rollen und Kompetenzen über Abteilungsgrenzen hinweg, innerhalb eines DevOps Teams, ist ständige Kommunikation und Abstimmung nötig. Dabei sollen nicht nur Ergebnisse und Daten, sondern auch Analysen und Ideen ausgetauscht werden. Es geht um den freien Austausch von Informationen sowie einen Prozess zum Wissensmanagement.

Diese Prinzipien sind, je nach Fokus eines Unternehmens, unterschiedlich wichtig – beispielsweise profitiert nicht jedes Unterhemen von einer Automatisierungspipeline, welche die Basis für mehrere Releases pro Tag legt. Werden nur wenige Releases pro Monat benötigt, stehen die Kosten für die Umsetzung dieser Lösung nicht im Verhältnis zum Nutzen, welcher daraus generiert wird. Der Begriff DevOps umfasst zwar speziell die Bereiche Entwicklung und Betrieb, tatsächlich kann das Konzept jedoch auf das gesamte Unternehmen ausgerichtet werden

– wenn diese Kultur in der gesamten Organisation verwurzelt ist, ist DevOps auch am effizientesten (RedHat, 2019).

Abbildung 11 veranschaulicht die Positionierung von DevOps – es kann als Ziel betrachtet werden, welches nach festgelegten Leitkriterien (WAS?) definiert wird. Agile Entwicklung, Continuous Integration (CI) und Continuous Delivery (CD) hingegen sind Prozesse, Methoden und Tools, welche das Fundament zu einer nachhaltigen DevOps Implementierung legen (Dawson, 2016).

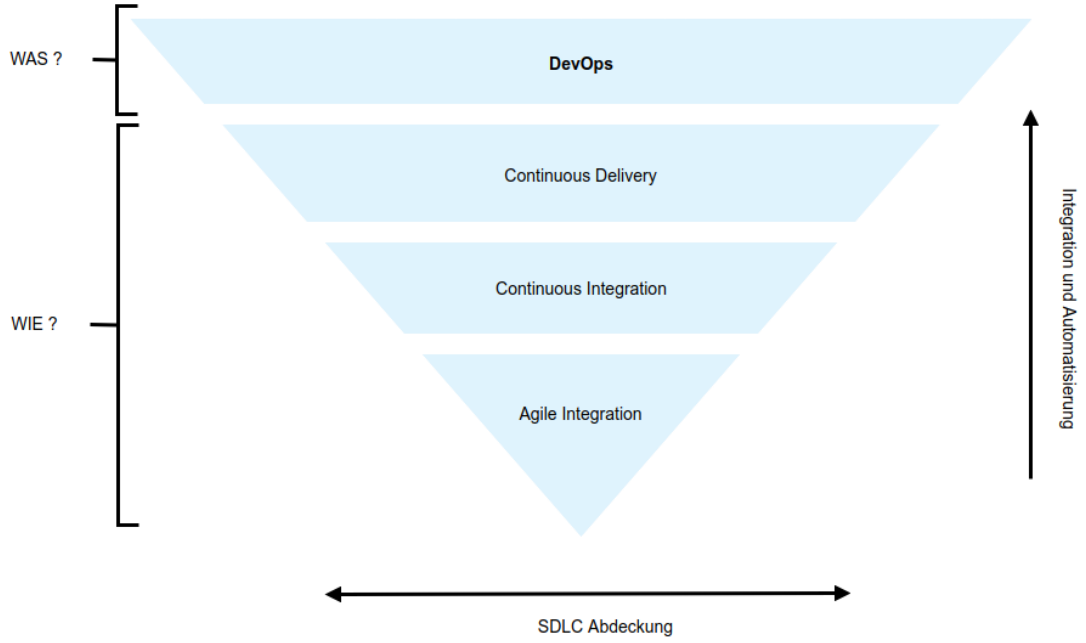

Abbildung 11: Positionierung von DevOps (eigene Darstellung in Anlehnung an Dawson, 2016)

Continuous Integration bedeutet, dass neue Funktionen bzw. Programmteile automatisiert getestet und zusammengeführt[8] werden, anstatt das beispielsweise nur einmal täglich zu definierten Zeiten zu erledigen (typischerweise einmal nachts durch einen so genannten „Nightly Build“). Der Vorteil dabei ist, dass Fehler frühzeitig erkannt werden können und somit die Softwarequalität gesteigert wird. Das Ziel von CI ist es, dass die Anwendungen zu jedem Zeitpunkt in einem funktionstüchtigen Zustand ist (Humble & Farley, 2011). Der CI Vorgang wird dabei automatisiert ausgelöst, indem die neu entwickelten Funktionalitäten in das Versionsverwaltungssystem eingecheckt[9] werden.

Continuous Delivery hingegen erweitert CI und stellt sicher, dass die zuvor getestete Softwareversion auf Knopfdruck bereit zum Deployen ist. Das beinhaltet beispielsweise das automatisierte Erstellen des Paketes und das Bereithalten dieses Paketes im Repository. Ziel

[8] Zusammenführen oder Mergen bedeutet das Integrieren des abgezweigten Codestandes in den Main-Branch eines Repositories

[9] Einchecken oder auch Committen ist das Übertragen einer lokalen Arbeitskopie in das zentrale Projektrepository.

von CD ist eine Codebasis verfügbar zu halten, welche jederzeit für die Implementierung auf einer Produktionsumgebung bereit ist (RedHat, 2019).

Abbildung 12 verdeutlicht den Unterschied zwischen CI und CD:

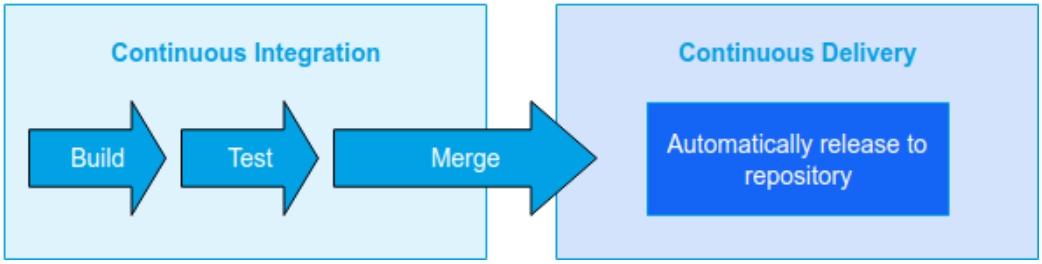

Abbildung 12: CI / CD (eigene Darstellung in Anlehnung an RedHat, 2019)

Mittels DevOps können zusammenfassend also die Bereiche Softwareentwicklung und Betrieb miteinander verschmolzen werden, um durch Automatisierung, Wissensmanagement und einer verbesserten Kultur eine reibungslosere und effektivere Zusammenarbeit über Abteilungsgrenzen hinweg zu gewährleisten. Was in diesem Konstrukt jedoch noch fehlt, ist die Integration des Fachbereiches bzw. des Kunden, für den bzw. mit dem die betreffende Anwendung entwickelt wird. Speziell in Anbetracht von Low-Code Entwicklungsplattformen, mit denen, wie bereits erwähnt, Citizen Developer die Möglichkeit haben, direkt in den Fachabteilungen Anwendungen mit Hilfe der IT-Abteilung zu entwickeln, ist eine enge und laufende Abstimmung auch in diese Richtung nötig. Entwicklung, Betrieb, Fachabteilung und Management müssen gemeinsam wertschöpfend an einer Anwendung arbeiten.

Um dieses Problem zu adressieren, wurde mit BizDevOps ein Konzept entwickelt, welches die Zusammenarbeit, die Organisation und die Abläufe von Unternehmensbereichen dahingehend optimiert, um schneller wirtschaftlichen Nutzen zu generieren (Schulze & Bo Tiedemann, 2018). Dieser Ansatz wird im folgenden Kapitel kurz erläutert.

2.4.1 **BizDevOps**

BizDevOps ist eine konsequente Weiterentwicklung des DevOps Ansatzes und beseitigt den letzten noch bestehende funktionalen Silo, indem Stakeholder des betreffenden Geschäftsbereiches ebenfalls in den Entwicklungs- und Betriebsprozess der Anwendung mit integriert werden. Es entsteht ein abteilungsübergreifendes und interdisziplinäres Team, welches ein gemeinsames Ziel verfolgt und gemeinschaftlich für den Projekterfolg verantwortlich ist. Durch das direkte Einbeziehen des Kunden in den Entwicklungsprozess, können dessen Rückmeldungen nahezu ohne Zeitverlust umgesetzt werden. Einzelne Personen können mehrere Rollen innerhalb eines BizDevOps Teams einnehmen.

Der Ablauf eines BizDevOps Zyklus ist in Abbildung 13 ersichtlich:

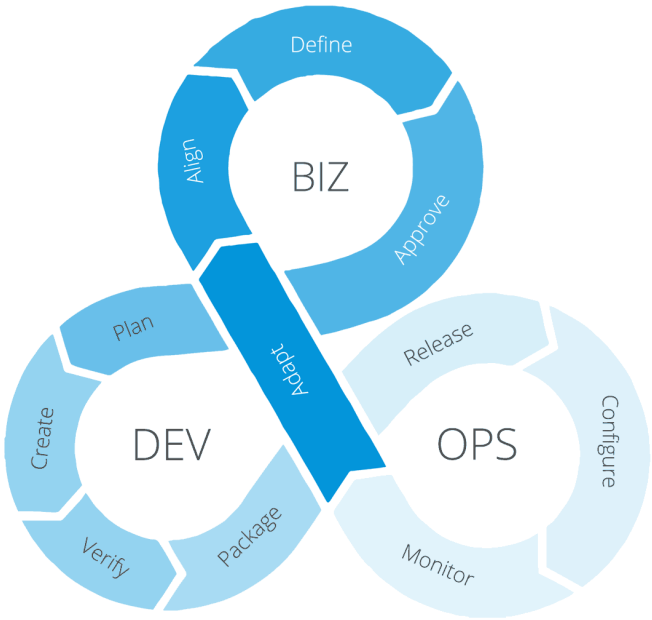

Abbildung 13: BizDevOps Zyklus (eigene Darstellung in Anlehnung an Blueprint, 2020)

Die Schritte im BizDevOps Zyklus, entsprechen den in Kapitel 2.3 beschriebenen Punkten des SDLC. Jedoch sind diese einzelnen Schritte im BizDevOps Zyklus granularer aufgeteilt und bestimmten Rollen zugeordnet. Beispielsweise besteht der SDLC Punkt „Operation“ aus den BizDevOps Schritten „Release“, „Configure“ und „Monitor“. Ein einzelnes, autonom agierendes, interdisziplinäres Team ist verantwortlich für den kompletten Anwendungslebenszyklus. Von der Planung über die Umsetzung bis hin zum Betrieb und der Wartung. Gegenseitige Schuldzuweisungen zwischen Geschäftsbereich und der IT-Abteilung im Falle eines Projektmisserfolgs sowie das „über den Zaun werfen“ von neuen Anforderungen seitens des Fachbereichs an das Entwicklungsteam, können mit diesem Ansatz vermieden werden. Diese enge, bereichsübergreifende Zusammenarbeit, ist jedoch für viele Unternehmen ungewohnt und entsteht nicht von heute auf morgen. Diese „neue Art“ der Zusammenarbeit muss in den Köpfen der MitarbeiterInnen sowie in der Unternehmenskultur verankert werden.

Abschließend wird in Tabelle 1 noch eine Abgrenzung der in den letzten Kapiteln beschriebenen Konzepte, Prozesse und Werkzeuge veranschaulicht:

Agile Methoden	CI / CD	DevOps / BizDevOps
Fokussiert auf *Entwicklungsprozess*	Fokussiert auf *Software-Lebenszyklus*	Fokussiert auf *Kultur, Werte, Kommunikation*
im Mittelpunkt steht *ständige Veränderung*	im Mittelpunkt stehen *Werkzeuge*	im Mittelpunkt stehen *Rollen & Verantwortlichkeiten*
welche die *Auslieferung* beschleunigt	welche die *Automatisierung* vorantreiben	welche die *Reaktionsfähigkeit* verbessern

Tabelle 1: Abgrenzung Agil, CI / CD, DevOps / BizDevOps (eigene Darstellung in Anlehnung an Steven, 2018)

2.5 Zusammenarbeit in international verteilten Teams

Ein fundamentaler und kritischer Faktor für den Erfolg / Misserfolg eines Softwareprojektes sind die Leistungen bzw. die Anstrengungen der einzelnen Teammitglieder (Acuña, Gómez, & Juristo, 2008). Heutzutage leben wir in einer globalisierten Welt, in der es normal ist, mit KollegInnen in einem Projekt virtuell zusammenzuarbeiten, welche in verschiedenen Ländern und unterschiedlichen Zeitzonen sitzen. Die Zusammenarbeit in diesen virtuellen Teams bringt spezielle Herausforderungen mit sich, beispielsweise erhöht sich durch die fehlende direkte Kommunikation von Angesicht zu Angesicht die Gefahr, dass Missverständnisse entstehen (Gibson & Cohen, 2003), woraus sich schneller Konflikte ergeben können als in Teams, welche im selben Raum sitzen (Hinds & Bailey, 2003). Einer der wichtigsten Faktoren, um als Team in solch einem Setting erfolgreich zu sein, ist daher die effektive Zusammenarbeit zwischen den verteilten Teammitgliedern (Hansen, 2009). Welche Punkte diese Zusammenarbeit beeinflussen sowie welche Probleme entstehen können, soll in den folgenden Kapiteln beleuchtet werden.

2.5.1 Herausforderungen in virtuellen Teams

Obwohl es heutzutage durch die technischen Hilfsmittel mit wenigen Mausklicks möglich ist, den/ die gewünschte(n) KollegIn per Online Meeting zu erreichen, stellt die räumliche Entfernung trotzdem eine Barriere für die Kommunikation, die Zusammenarbeit sowie den Aufbau von Beziehungen zwischen den Teammitgliedern dar. Werkzeuge, welche die verteilten Zusammenarbeit unterstützen sollen, können für ein Unternehmen auch zu einer „Belastung“ werden, wenn MitarbeiterInnen diese bis zu einem Punkt „überbeanspruchen“, an dem der eigentliche Nutzen dieser Werkzeuge verloren geht (Feldman & Orlikowski, 2011).

Informelle, oft zufällig stattfindende Zusammentreffen, beispielsweise in der Kaffeeküche, sind nicht möglich, wodurch der ungezwungene Austausch von z.B. Wissen oder der Aufbau einer Beziehung bzw. einer Vertrauensbasis erschwert wird. Ein weiterer Punkt, welcher die Zusammenarbeit negativ beeinflussen kann, sind Statusunterschiede - so sind Teammitglieder, welche höhere Positionen bekleiden und in „Machtzentren“, beispielsweise dem Firmenhauptsitz, angesiedelt sind, weniger gewillt, ihr Wissen mit KollegInnen aus anderen Niederlassungen zu teilen (Gumienny, Gericke, Wenzel, & Meinel, 2013). Eine weitere Herausforderung ist es, Gruppenbildungen innerhalb der Teams zu vermeiden. Sind beispielsweise mehrere

Teammitglieder sowie der Projektleiter im Firmenhauptsitz angesiedelt, kann es leicht passieren, dass Entscheidungen innerhalb dieser Gruppe getroffen werden, ohne die Mitglieder, welche an anderen Standorten arbeiten, miteinzubeziehen (Karis, Wildman, & Mane, 2014). Dadurch kann eine „wir-gegen-die“ Mentalität entstehen, welche für eine erfolgreiche Zusammenarbeit im Team nicht förderlich ist. Dieses Verhalten hat unter anderm mit Vertrauen zu tun, das in verteilten Teams zwar hergestellt werden kann, durch die oben genannten Faktoren jedoch längere Zeit beansprucht, als in traditionellen Teams.

Weitere Hindernisse können sich durch Sprachbarrieren oder kulturelle Differenzen ergeben. Handlungen bzw. Aussagen, welche zum Beispiel für ÖsterreicherInnen ganz normal erscheinen, können unter Umständen in asiatischen Ländern Unverständnis hervorrufen oder zu Missverständnissen führen. Im schlimmsten Fall kann dadurch ein Projekt gefährdet werden, da ProjektmitarbeiterInnen nicht mehr zusammenarbeiten wollen.

Abhängig davon, in welchen Ländern die einzelnen Teammitglieder beheimatet sind, kann auch die Zeitdifferenz zu einem Problem werden. Große Unterschiede der Zeitzone von 8 Stunden und mehr, können ein Hindernis bei der Koordination bzw. Absprache der Teammitglieder untereinander darstellen, was eine effektive Kommunikation im Team deutlich erschwert. Dieses Problem gilt ebenso für regelmäßige Meetings, welche abhängig von der Zeitzone für bestimmte Teammitglieder zu für diese ungünstigen Zeiten stattfinden können. In solchen virtuellen Meetings ist es auch einfacher möglich, im Gegensatz zu Besprechungen an denen die Personen physisch teilnehmen, mit den eigenen Gedanken und Taten nicht bei der derzeit besprochenen Thematik zu sein und sich stattdessen mit anderen Dingen zu beschäftigen – die Teilnahme kann einfacher vorgetäuscht werden (Leonardi, Treem, & Jackson, 2010).

2.5.2 Faktoren für effektive Zusammenarbeit

Nachdem im vorherigen Kapitel die Herausforderungen besprochen wurden, welche durch die Zusammenarbeit in verteilten Teams entstehen können, soll in diesem Kapitel ein kurzer Überblick über den derzeitigen Stand der Forschung zu diesem Thema gegeben werden. Welche Faktoren sind besonders zu berücksichtigen, um ein effektives Teamwork in global verteilten Teams zu gewährleisten? Laut Gurram und Bandi (2012), die im Zuge ihrer Arbeit eine systematische Literaturrecherche durchgeführt haben, bei der insgesamt 45 Studien aus den Jahren 2001 bis 2012 analysiert und zusammengefasst wurden, sind Einflussfaktoren, welche in Tabelle 2 ersichtlich sind, besonders zu berücksichtigen, um in einem verteilten Team effektive Zusammenarbeit sicherzustellen. Die Reihenfolge ist dabei nicht relevant.

Faktor	Definition
Kommunikation	Umfasst den Austausch von Informationen zwischen zwei oder mehreren Teammitgliedern unter Verwendung der entsprechenden Terminologie sowie auf die vorgeschriebene Art und Weise. Bildet das Herzstück der Zusammenarbeit, da es ein gemeinsames Verständnis bezüglich der eingesetzten Prozesse sowie der zu verfolgenden Ziele erzeugt. Dieser

	Punkt beinhaltet ebenso informelle, nicht das Projekt betreffende, Kommunikation.
Teamorientierung	Bezieht sich auf die Überzeugung, dass Teamziele wichtiger sind als die Ziele einzelner Personen und die Fähigkeit, das Verhalten anderer bei Interaktionen innerhalb der Gruppe zu berücksichtigen und wertzuschätzen. Spiegelt die Akzeptanz der Team-Normen, das Zusammengehörigkeitsgefühl sowie Bedeutung der Teammitgliedschaft wider.
Vertrauen	Betrifft die gemeinsame Überzeugung, dass alle Teammitglieder Handlungen, welche a priori vereinbart wurden, zufriedenstellend ausführen sowie die Interessen anderer Teammitglieder durch ihre Handlungen schützen. Vertrauen ist ein essenzieller Faktor, welcher Personen aus unterschiedlichen Regionen, mit unterschiedlichen Arbeitsweisen und verschiedenen Kulturen als Team agieren lässt.
Kultur	Kultur ist eine Reihe gemeinsamer Erfahrungen, Erkenntnisse und Meinungen, welche Mitglieder einer Gruppe, einer Organisation oder eines Landes teilen und auf Basis derer, Handlungen ausgeführt werden.
Koordination	Koordination bezieht auf die Aufgaben einzelner Teammitglieder, welche rechtzeitig und umfassend erledigt werden müssen. Durch ggf. auftretende Abhängigkeiten zwischen den Aufgaben, kommt dem Punkt Koordination besondere Bedeutung zu, da die Performance (beispielsweise durch Stillstand) des gesamten Teams dadurch beeinflusst werden kann.
Vertrautheit	Bezieht sich einerseits auf das Wissen über die anderen Mitglieder im Team sowie andererseits auf das Wissen über die zu erledigenden Aufgaben. Vertrautheit zwischen den einzelnen Teammitgliedern erleichtert die Abstimmung und Interaktion zwischen diesen, beispielweise durch das Wissen, wer für welche Angelegenheiten der korrekte Ansprechpartner ist und wie mit der entsprechenden Person umgegangen werden soll.
Kommunikations-technologie	Bezeichnet die Verwendung von synchronen sowie asynchronen Technologien, um die Kommunikation zwischen den einzelnen Teammitgliedern zu erleichtern. Die eingesetzte Technologie sollte einfach zu bedienen sein und möglichst viele Kommunikationsbedürfnisse abdecken.
Geteilte Führung	Aufteilen von Verantwortung und Führungsaufgaben auf mehrere Teammitglieder anhand von deren Kenntnissen und Fähigkeiten. Das beinhaltet das Planen und Zuweisen von Aufgaben, Motivation der Teammitglieder sowie Hilfe bei Problemen.

Feedback	Feedback umfasst sowohl das Geben als auch das Einholen von Feedback bezüglich des eingesetzten Vorgehensmodells, der umgesetzten und geplanten Aufgaben, der Verfügbarkeit von Werkzeugen sowie der Qualität der Anwendung als Ganzes. Betrifft das Akzeptieren von positiven sowie auch negativen Rückmeldungen. Gezieltes Feedback minimiert frühzeitig das Risiko von Fehlentwicklungen.
Veränderung	Ist die Fähigkeit der einzelnen Teammitglieder, Änderungen in der Team- bzw. Projektumgebung zeitnah zu identifizieren und rechtzeitig geeignete Maßnahmen in die Wege zu leiten.
Vertretung	Dieser Punkt beinhaltet die Bereitschaft sowie Fähigkeit, die Aufgaben anderer Teammitglieder bei Bedarf zu übernehmen bzw. bei deren Umsetzung zu unterstützen. Das kann beispielsweise der Fall sein, wenn einzelne Teammitglieder durch zu viele Aufgaben überlastet oder Ausfälle durch Krankheit zu verzeichnen sind.

Tabelle 2: Faktoren für effektive Zusammenarbeit in verteilten Teams (Gurram & Bandi, 2012)

Zwischen deinen einzelnen Faktoren bestehen zum Teil Verbindungen, so haben beispielsweise Zhu und Lee (2017) in ihrer Publikation „*Global virtual team performance, shared leadership, and trust: proposing a conceptual framework*“ einen Zusammenhang zwischen den Faktoren „Geteilter Führung“ und „Vertrauen“ festgestellt, welche wiederrum direkt die Leistung in einem verteilten Team beeinflussen. Geteilte Führung kann laut Zhu und Lee (2017) erst über einen längeren Zeitraum der Zusammenarbeit im Team entstehen, Voraussetzung dafür ist bereits existierendes Vertrauen zwischen den Teammitgliedern. „Vertrauen“ wiederrum ist stark an den Faktor „Vertrautheit“ gekoppelt. Weitere Verbindungen bestehen beispielsweise zwischen den Faktoren „Vertretung“ sowie „Team Orientierung“ und „Koordination“. Es ist also notwendig, nicht nur einzelne Faktoren zu optimieren, sondern auch deren Zusammenspiel zu berücksichtigen.

Abschließend werden in Tabelle 3 noch auszugsweise Strategien genannt, welche laut Gurram und Bandi (2012) die oben genannten Faktoren gezielt adressieren und Möglichkeiten bieten, diese Bereiche zu optimieren.

Faktor	Strategie
Kommunikation	• Teilen aller Besprechungsnotizen mit sämtlichen Teammitgliedern • Ernennen eines „Kommunikationsbeauftragten“ welcher alle Teammitglieder, die eine bestimmte Kommunikation versäumt haben, informiert • In regelmäßigen Meetings (beispielsweise Daily Standup) werden nur wichtige Angelegenheiten besprochen, um die Aufmerksamkeit aller Teilnehmer zu gewährleisten

	• Persönliche Kalender (beispielsweise Outlook) sollten allen Teammitgliedern lesbar zugänglich sein • Coachings, um direkte und ehrliche Kommunikation zu gewährleisten • Etablieren von bevorzugten Kommunikationswegen und Eskalationsstufen (bei nicht Erreichbarkeit), um mit einzelnen Teammitgliedern in Kontakt zu treten
Vertrauen	• Zusammenbringen der Teammitglieder an einem zentralen Ort zu Beginn des Projektes • Fix eingeplante Zeit für Smalltalk bevor Besprechungen beginnen, lockere Kommunikation • Gemeinsames Feiern von Erfolgen • Offene Kommunikation, alle Ideen sind Willkommen • Wichtige Änderungen werden im gesamten Team besprochen • So oft als möglich an einem Ort zusammenkommen • Online-Teamevents • Erstellen einer Teamseite mit Informationen und Bildern der einzelnen Mitglieder
Vertrautheit	• Onboarding Maßnahmen bei denen die verfügbaren Werkzeuge, Rollen und Aufgaben besprochen werden • Projekt Kick-Off Event für alle Teammitglieder
Kommunikationstechnologie	• Einsetzen von Videokonferenz Lösungen bzw. aktivieren der Webcam (Herstellen von „Augenkontakt“, erfassen der Körpersprache) • Bereitstellen von Lösungen, um technisches und organisatorisches Projektwissen zu teilen und zu verwalten
Geteilte Führung	• An jedem Standort ist ein gleich großes Team mit ähnlichen Fähigkeiten vorhanden, somit können Führungsaufgaben einfacher geteilt werden
Feedback	• Fix eingeplante Feedback Meetings (beispielsweise Retrospektiven) • Direktes und offenes Feedback zu jeder Zeit • Whole-Team-Approach
Vertretung	• Aktiver Wissensaustausch zwischen allen Teammitgliedern • Herausheben der gemeinsamen Ziele und der gemeinsamen Verantwortung für den Projekterfolg

Tabelle 3: Strategien zur effektiven Zusammenarbeit (Gurram & Bandi, 2012)

In der Praxis sind nicht alle der oben genannten Strategien umsetzbar, da die Rahmenbedingungen dafür nicht immer gegeben sind. Beispielsweise ist es oft nicht möglich, die Teams gleichermaßen auf mehrere Standorte zu verteilen oder es ist auf finanziellen Gründen nicht möglich, das alle ProjektmitarbeiterInnen zu Projektbeginn an einem zentralen Standort zusammengebracht werden.

3 METHODEN

Um die in Kapitel 1.1 gestellte Forschungsfrage dieser Masterarbeit beantworten, und die damit verbundene Hypothese verifizieren bzw. falsifizieren zu können, wird ein Methodenmix bestehend aus Fallstudie und ExpertInneninterview angewandt. Das im Zuge dieser Masterarbeit entwickelte Vorgehensmodell (siehe Kapitel 4.1) wird dazu in einer Fallstudie innerhalb der ANDRITZ AG einem Praxistest unterzogen. Anschließend an diese Fallstudie wird mittels ExpertInneninterviews überprüft, ob sich das erstellte Modell für den praktischen Einsatz eignet oder ob Optimierungsmaßnahmen ergriffen werden müssen.

In den folgenden Kapiteln werden die Fallstudie sowie die daran anschließend durchgeführten ExpertInneninterviews genauer erläutert.

3.1 Fallstudie

Fallstudien zählen zu den qualitativen, empirischen Forschungsmethoden und untersuchen ein zeitgenössisches Phänomen in einem realweltlichen Kontext, bei dem die Grenze zwischen beobachtetem Phänomen und Kontext nicht klar erkennbar ist (Yin, 2009). Der Kontext bzw. die Rahmenbedingungen sind dabei ein wichtiger Bestandteil der Studie, da darin wertvolle Informationen vermutet werden. Durch Fallstudien ist es möglich, Prozessabläufe, Entwicklungen sowie Ursache- und Wirkungszusammenhänge nachzuvollziehen und dadurch in der Praxis relevante, datenbasierte Aussagen treffen zu können (Albers, Konradt, Walter, Wolf, & Klapper, 2007)

Laut Yin (2009) können drei Arten von Fallstudien unterschieden werden, welche sich jeweils anhand der Anzahl der untersuchten Fälle sowie der Anzahl der betrachteten Analyseeinheiten differenzieren lassen. Diese unterschiedlichen Arten sind in Abbildung 14 ersichtlich.

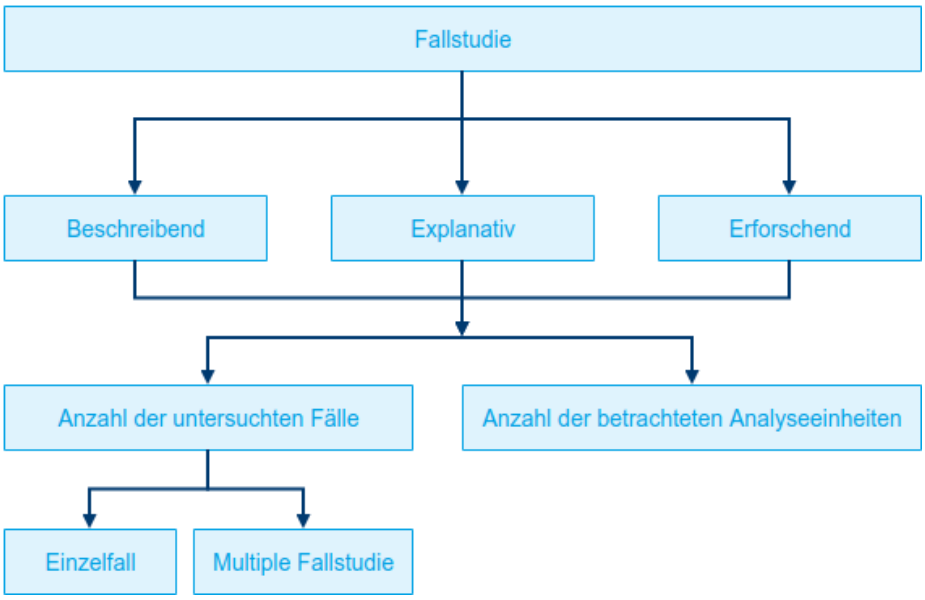

Abbildung 14: Arten von Fallstudien (eigene Darstellung in Anlehnung an Yin, 2009)

1. Beschreibende oder deskriptive Fallstudien dienen der Darstellung eines in der Theorie beschriebenen Sachverhaltes durch die komplette Erfassung und Beschreibung eines Phänomens sowie dessen spezifischen Kontextes (Yin, 2009). Es werden hier jedoch keine Erklärungen für die beobachteten Phänomene entwickelt (Lamker, 2014).
2. Explanative Fallstudien erklären und begründen praktische Erfahrungen anhand theoretischer Aussagen, welche vorab formuliert wurden (Yin, 2009).
3. Erforschende oder explorative Fallstudien versuchen Hypothesen, welche vorab im Rahmen einer Theorie entwickelt wurden, näher zu bestimmen (Yin, 2009).

Weiters kann anhand der untersuchten Fälle unterschieden werden. Multiple Fallstudien gelten als Aussagekräftiger als Einzelfallstudien, da die gewonnenen Erkenntnisse durch gezielte Vergleiche kritisch beleuchtet werden können. Der Nachteil von multiplen Fallstudien im Vergleich zu Einzelfallstudien ist jedoch ein höherer Zeit- sowie Ressourcenaufwand – außerdem kann es vorkommen, das besonders kritische und spezifische Fälle oftmals nur einmalig auftreten und daher multiple Fallstudien nicht möglich sind (Yin, 2009). Des Weiteren können innerhalb einer Fallstudie eine oder mehrere Analyseeinheiten betrachtet werden – Beispiele für solche Analyseeinheiten sind unter anderem Prozesse oder Funktionsbereiche innerhalb einer Organisation.

Im Zuge dieser Arbeit wird eine explanative Einzelfallstudie angewandt welche ein Artefakt, das Vorgehensmodell, untersucht. Als Datenerhebungsmethode werden im Anschluss an die Fallstudie ExpertInneninterviews durchgeführt.

Im Folgenden wird der praktische Einsatz des erstellten Vorgehensmodells innerhalb der ANDRITZ AG im Zuge der Fallstudie sowie dessen Rahmenbedingungen beschrieben. Nach einer kurzen Vorstellung des Unternehmens folgt eine Beschreibung des Projektes. Der genaue Ablauf der Fallstudie ist in Kapitel 4.3 beschrieben.

3.1.1 **Das Unternehmen**

ANDRITZ ist ein internationaler Technologiekonzern welcher Anlagen, Systeme, Ausrüstungen und Serviceleistungen für Wasserkraftwerke, die Zellstoff- und Papierindustrie, die Stahl- und metallverarbeitende Industrie sowie die kommunale und industrielle Fest-Flüssig Trennung anbietet (ANDRITZ AG, 2020).

Der Hauptsitz des börsennotierten Konzerns befindet sich in Graz, Österreich. Weltweit werden in über 280 Standorten, welche auf mehr als 40 Ländern verteilt sind, etwa 29.700 MitarbeiterInnen beschäftigt - davon rund 3.300 in Österreich. Der Umsatz im Jahr 2018 betrug 6,0315 Milliarden Euro. Entwickelt hat sich das Unternehmen aus einer Eisengießerei, welche 1852 vom Unternehmer Josef Körösi im Grazer Vorort Andritz gegründet wurde.

Die ANDRITZ Gruppe teilt sich in folgende Geschäftsbereiche auf:

- ANDRITZ Hydro – Wie der Name bereits vermuten lässt, werden in diesem Bereich hauptsächlich elektromechanische Ausrüstungen für Wasserkraftwerke wie beispielsweise Turbinen oder Generatoren hergestellt. Des Weiteren stehen

verschiedenen Pumpen und Turbogeneratoren auf der Produktpalette. Der Anteil der Sparte Hydro am gesamten Auftragseingang der ANDRITZ Gruppe im Jahr 2018 betrug 22%.

- ANDRITZ Pulp & Paper – Mit einem Anteil von rund 39% am gesamten Auftragseingang der ANDRITZ Gruppe im Jahr 2018 ein relativ großer Bereich – es werden Anlagen für die Erzeugung aller Arten von Faserstoffen, Papier, Tissuepapier und Karton, sowie Kessel für die Energieerzeugung angeboten.
- ANDRITZ Metals – Mit dem Kauf (95% der Anteile) der Schuler AG im Jahr 2013 wurde der Geschäftsbereich Metals um Pressen sowie Pressenlinien für die Metallumformung erweitert. Außerdem stehen Anlagen zur Herstellung von Bändern aus Edelstahl, Kohlenstoffstahl und Nicht-Eisen-Metallen sowie Industrieofenanalagen im Angebot. Die Sparte Metals hat im Geschäftsjahr 2018 29% des gesamten Auftragseingangs zu verantworten.
- ANDRITZ Seperation – Im Bereich Seperation werden Anlagen zur kommunalen und industriellen Fest-Flüssig-Trennung sowie zur Herstellung von Tierfutter- und Biomassepellets produziert.

Ein weiteres wichtiges Geschäftsfeld ist die Automatisierung. Hier werden Produkte und Dienstleistungen für die verschiedenen Anlagen angeboten mit dem Ziel, die industriellen Abläufe zu optimieren. Unter der Marke Metris bietet ANDRITZ eine breite Palette an innovativen digitalen Produkten und Dienstleistungen im Bereich Industrial Internet of Things (IIoT) an (ANDRITZ AG, 2020), welche angepasst auf die jeweiligen Kundenbedürfnisse die Effizienz von Anlagen oder den Einsatz von Ressourcen optimieren können. Des Weiteren können die Anlagen im laufenden Betrieb überwacht und somit Produktionsausfälle vermieden werden.

3.1.2 **Projektbeschreibung**

Der Zweck der im Zuge dieser Fallstudie entwickelten Anwendung, ist das Verwalten von Berechnungsdaten (Design Certification) für verschiedene, innerhalb der ANDRITZ Gruppe gefertigter, Maschinen. Die Kalkulationsdaten können dabei vom Berechnungsingenieur auf Komponentenebene eigegeben werden. Eine Maschine wird anhand einer Baumstruktur, welche sich durch den Import einer Bill of Material[10] (BOM) ergibt, abgebildet. Solche BOMs bestehen aus einer hierarchischen Darstellung der kompletten Maschine – Level 0 ist beispielsweise die Maschine selbst, Level 1 sind so genannte Sub-Assemblies, d.h. größere Bauteile, welche aus mehreren Einzelbauteilen bestehen. Level 2 sind die einzelnen Bauteile selbst.

Die notwendigen Daten kommen direkt aus dem ANDRITZ SAP-System. Dafür war es notwendig, eine Schnittstelle zwischen der in der SAP-Cloud betriebenen Mendix Anwendung und dem on-

[10] BOM bezeichnet eine strukturiere Anordnung von Bauteilen einer Maschine aufgeteilt in Levels – beispielsweise ist Level 0 die komplette Maschine, welche wiederrum n Level 1 Elemente (z.B. Bauteile) enthalten kann. Diese Bauteile enthalten wiederrum n Level 2 Elemente (einzelne Komponenten) usw.

premises betriebenen ANDRITZ SAP-Systems, einzurichten. Um dies zu ermöglichen, wurde der SAP-Cloud Connector verwendet, welcher eine sichere Verbindung von Anwendungen außerhalb des ANDRITZ Netzwerkes zu Systemen innerhalb der ANDRITZ Infrastruktur etabliert. Abbildung 15 zeigt die erwähnte Baumstruktur, welche die Maschine sowie deren Bauteile übersichtlich abbildet.

Abbildung 15: Darstellung einer Maschine sowie der verwendeten Komponenten (eigene Darstellung)

Der Berechnungsingenieur hat die Möglichkeit, nachdem die Maschinendaten (Umdrehungen pro Minute, Drehmomentfaktor usw.) sowie die BOM-Struktur aus dem SAP-System importiert wurden, eigens erstellte Komponenten (diese beinhalten die benötigen Attribute, welche die Berechnungsdaten speichern) einzelnen Bauteilen oder Sub-Assemblies zuzuweisen. Diese Komponenten sind als so genannte Blueprints in der Anwendung hinterlegt und können beliebig angepasst werden. Anhand der darin eingebenden Werte, werden automatisch die für die Qualitätssicherung notwendige Parameter berechnet. Änderungen (beispielsweise das hinzufügen oder entfernen von Attributen, ändern der verwendeten Einheiten usw.) in diesen Blueprints, münden dabei in einer neuen Version dieser, wodurch bereits verwendete und zugewiesene Blueprints und die darauf basierenden und automatisch berechneten Parameter, unverändert bleiben. Ebenso ist es möglich, bereits mit Daten befüllte Blueprints, die einer anderen (baugleichen) Maschine zugewiesen sind, zu übernehmen.

Der Status, in dem sich eine Komponente befindet, wird anhand eines grafischen Indikators in der Baumstruktur, wie in Abbildung 15 dargestellt, abgebildet. Somit ist einfach und schnell ersichtlich, welche Komponenten noch Eingaben benötigen, welche bereits mit Daten versorgt aber noch nicht freigegeben wurden und schließlich, welche Komponenten abgeschlossen und somit freigegeben sind. Der jeweils „niedrigste“ Status wird dabei von Einzelkomponentenebene über die Sub-Assemblies bis hin zur Maschine (Level 0) weitergegeben.

Wie die Blueprints, sind auch die BOMs selbst versioniert. Das bedeutet, wird eine neue Version der Maschine aus SAP importiert, wird automatisch eine neue Version der Baumstruktur erstellt. Dabei kann optional gewählt werden, ob die bereits hinterlegten Berechnungsdaten aus der „alten“ Version übernommen werden sollen. Der Status der Komponenten, die übernommen werden, ändert sich dabei automatisch auf „Daten vorhanden jedoch noch nicht freigegeben“. Des Weiteren sind diverse Such- und Filteroptionen in der Anwendung enthalten, um Daten schnell und effektiv auffinden zu können.

Bei der Anwendung selbst handelt es sich um eine responsive Website, welche in der SAP-Cloud betrieben wird. Die Datenhaltung erfolgt in einer PostgreSQL Datenbank. Ein Backup dieser Datenbank erfolgt täglich via Skript, das automatisiert von einem Jenkins Build Job ausgeführt wird. Außerdem steht eine mobile Version, welche auf einem Apple iPad betrieben werden kann, zur Verfügung. Zweck dieser mobilen Variante ist es, dem Berechnungsingenieur die Möglichkeit zu geben, bei Bedarf mit all den notwendigen Daten direkt zur betreffenden Maschine zu gehen und dort ggf. notwendige Schritte durchzuführen.

Die Autorisierung und Authentifizierung erfolgt via Active Directory Federation Services (ADFS), somit können sich ANDRITZ MitarbeiterInnen ohne Eingabe von Username und Passwort direkt via Single Sign On (SSO) durch ihren ANDRITZ Account anmelden. Die Rechte innerhalb der Anwendung werden durch Usergruppen (AdministratorIn, PowerUser, BerechnungsingenieurIn) gesteuert. Derzeit benutzen rund 60 MitarbeiterInnen an verschiedenen Standorten diese im ANDRITZ RAD Service entwickelte Anwendung.

Der Ablauf der Fallstudie wird im Kapitel Ergebnisse (4.3) detailliert beschrieben.

3.2 ExpertInneninterview

Nachdem das Vorgehensmodell (siehe Kapitel 4.1) einem Praxistest (siehe Kapitel 4.3) unterzogen wurde, soll abschließend mittels ExpertInneninterviews evaluiert werden, ob sich das erstellte Modell für den praktischen Einsatz eignet und in welchen Punkten eventuell Nachbesserungen stattfinden müssen. Im Folgenden werden kurz die theoretischen Grundlagen sowie Kritikpunkte betreffend diese Forschungsmethode erläutert.

Ziel der ExpertInneninterviews ist es, zu einem Untersuchungsgegenstand passendes Fachwissen von Personen einzuholen, um es in weiterer Folge aufbereiten und danach nutzen zu können – es sollen neue Erkenntnisse gewonnen werden. Das ExpertInneninterview zählt zu den qualitativen Forschungsmethoden und ist innerhalb der Sozialwissenschaften aufgrund einer Debatte über die Objektivität dieser Methoden nicht unumstritten. Der Grund dafür ist die Gegenwart des / der InterviewerIn, denn es ist nicht auszuschließen, dass der / die Befragte auf Fragen eines anderen Gegenparts verschieden reagiert und geantwortet hätte (Mieg & Brunner, 2006). Abbildung 16 veranschaulicht dieses Problem. Das Interview ist eine soziale Interaktion und daher nie einfach auf Wissensaustausch beschränkt – die InteraktionspartnerInnen bringen Interessen und persönliche Probleme mit ein (Mieg & Brunner, 2006).

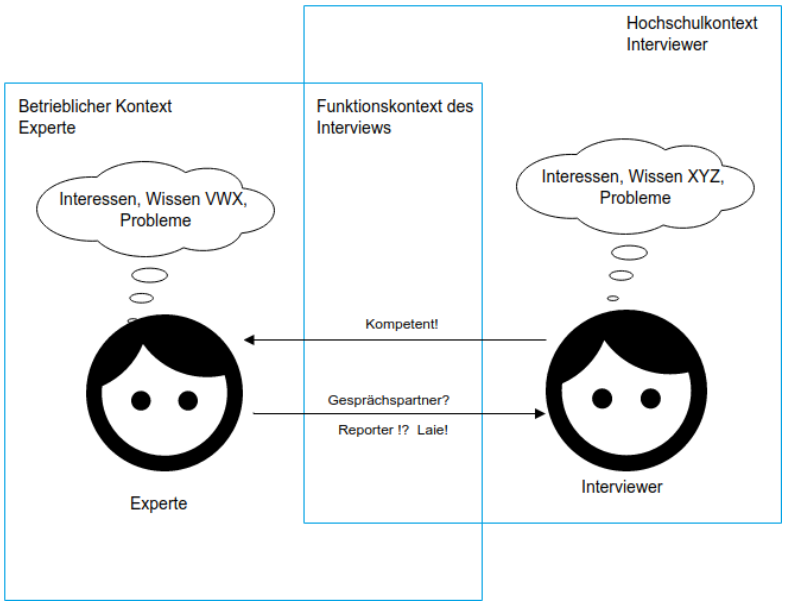

Abbildung 16: Interview als soziale Interaktion (eigene Darstellung in Anlehnung an Mieg & Brunner, 2001).

Um ein ExpertInneninterview durchzuführen, sollte der/die InterviewerIn als kompetente(r) GesprächspartnerIn auftreten und insbesondere mit der Fachsprache des / der ExpertIn vertraut sein (Mieg & Brunner, 2006). In dieser Arbeit wurde ein systematisierendes ExpertInneninterview angewandt, welches wie folgt charakterisiert werden kann:

- Systematisierende ExpertInneninterviews stellen das „aus der Praxis gewonnene, reflexiv verfügbare und spontan kommunizierbare Handlungs- und Erfahrungswissen“ (Bogner, Menz, & Littig, 2005) in den Mittelpunkt. Der / die ExpertIn soll dabei seine / ihre Sicht der Dinge bezüglich eines bestimmten Themenausschnittes darlegen und wird in erster Linie als „RatgeberIn“ gesehen, anstatt als eine Person, welche für den / die ForscherIn unzugängliches Wissen verfügt (Bogner, Menz, & Littig, 2005).

Die ExpertInnen im Kontext dieser Masterarbeit sind also KollegInnen des RAD Services und des Fachbereiches, welche direkt mit dem erstellten Vorgehensmodell im Zuge des Projektes, welches in Kapitel 3.1.2 beschrieben wurde, gearbeitet haben. Diese KollegInnen besitzen das relevante Fachwissen das Untersuchungsgebiet betreffend. Gläser und Laudel (2010) beschreiben ExpertInnen sowie das ExpertInneninterview wie folgt:

> *Experte beschreibt die spezifische Rolle des Interviewpartners als Quelle von Spezialwissen über die zu erforschenden Sachverhalte. Experteninterviews sind eine Methode, dieses Wissen zu erschließen* (Gläser & Laudel, 2010)

Als Basis für das folgende ExpertInneninterview wurde ein Interviewleitfaden erstellt, welcher dabei unterstützen soll, im Gespräch sämtliche Fragen vollständig und hinreichend zu beantworten (Mieg & Brunner, 2006). Dieser Leitfaden wird in Kapitel 3.2.2 vorgestellt. Die für das ExpertInneninterview notwendigen Vorbereitungsschritte sind im folgenden Kapitel ersichtlich.

3.2.1 **Vorbereitung auf das ExpertInneninterview**

Zur erfolgreichen Vorbereitung sowie Durchführung des ExpertInneninterviews, wurde die von Mieg & Brunner (2006) vorgeschlagene Vorgehensweise, ersichtlich in Abbildung 17, angewandt.

Ausgehend von der Forschungsfrage (siehe Kapitel 1.1) wurden Einflussgrößen identifiziert und darauf aufbauend ein ANDRITZ RAD Center of Excellence (siehe Kapitel 4.2) entworfen, welches individuell je nach Projekt sämtliche der identifizierten Faktoren beurteilen und darauf aufbauend Entscheidungen über die weitere Vorgehensweise (wie die zu verwendenden Umsetzungsmethode, Projektleitung usw.) treffen soll. Anhand des ExpertInneninterviews soll eruiert werden, ob das ANDRITZ RAD CoE und die darin bereitgestellten Methoden und Werkzeuge im praktischen Einsatz (siehe Kapitel 4.3) für die am Projekt beteiligten Personen mit deren spezifischen Rollen ausreichend waren. Diese Fragen wurden zur Übersichtlichkeit sowie einfacheren Auswertung in Kategorien eingeteilt. Die Wahl der ExpertInnen ist vom Untersuchungsgegenstand abhängig, im Fall dieser Masterarbeit wurden, wie bereits erwähnt, Personen aus dem ANDRITZ RAD Service sowie KollegInnen aus der Fachabteilung interviewt, welche direkt am umgesetzten Projekt beteiligt waren (siehe Kapitel 3.2.6). Sofern möglich, soll das Interview aufgezeichnet und visuelle Beobachtungen notiert werden, um eine Basis für die darauffolgende Transkription und Auswertung zu schaffen. Ist ein Aufzeichnen nicht möglich, besteht die Gefahr, dass wichtige Informationen verloren gehen. Zur Auswertung eines ExpertInneninterviews stehen unterschiedliche Methoden zur Verfügung. Die im Zuge dieser Arbeit angewandte Methode wird in Kapitel 3.2.4 beschrieben.

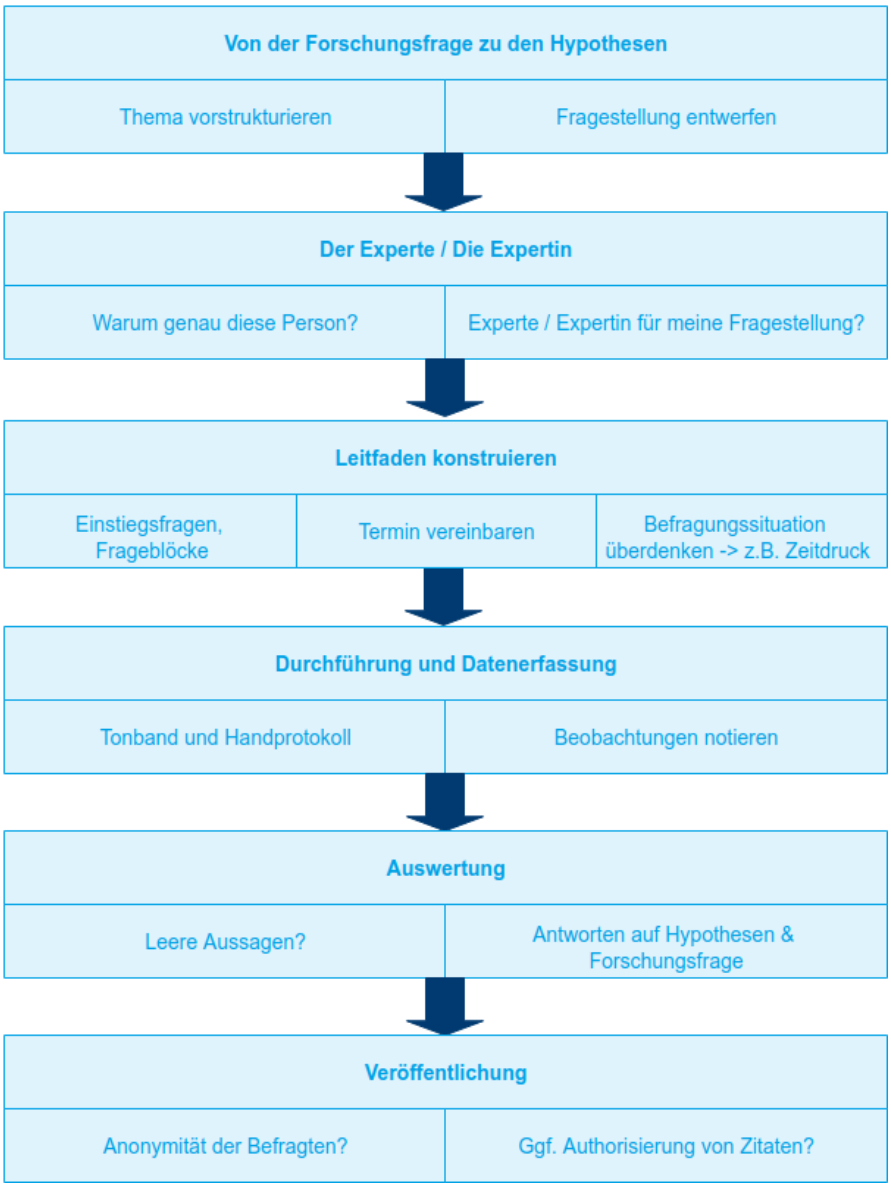

Abbildung 17: Vorgehensweise zum Durchführen eines ExpertInneninterviews (eigene Darstellung in Anlehnung an Mieg & Brunner, 2006)

3.2.2 Interviewleitfaden

Das Ziel dieses Interviewleitfadens ist es, Probleme und Herausforderungen, welche sich während der Umsetzung des Projektes ergeben haben, zu identifizieren. Es sollen Schwachstellen in den Methoden und Werkzeugen des ANDRITZ RAD CoE aufgedeckt und für zukünftige Projekte beseitigt werden. Durch die Rückmeldungen der ExpertInnen soll darüber hinaus die Hypothese, ob sich das erstellte Vorgehensmodell für einen praktischen Einsatz eignet, verifiziert bzw. falsifiziert werden. Der Leitfaden wird beim Leitfadeninterview nicht als starr angesehen, das bedeutet, weder die Formulierung noch die Abfolge unterliegen einer verbindlichen Reihenfolge – die Antwortmöglichkeiten sind vollkommen offen. Der Leitfaden dient in erster Linie als Orientierungsrahmen bzw. Gedächtnisstütze und soll unterstützend in Bezug auf die folgende Datenauswertung wirken.

Frage	Gesprächsimpuls	Weiterführende Fragen /Anmerkungen
	Kategorie: Gesprächseinstieg	
F1	Inwiefern haben Sie in der Vergangenheit bereits Erfahrungen bezüglich der Zusammenarbeit zwischen IT- und Fachabteilung sammeln können?	- positive/negative Aspekte?
	Kategorie: Anforderungsmanagement	
F2	Wenn Sie sich die erste Phase des Projektes, das Anforderungsmanagement, in Erinnerung rufen, – wie würden Sie diese Phase aus Ihrer Sicht beurteilen und zusammenfassen?	- positive/negative Aspekte? - Umfang angemessen? - Verbesserungsmöglichkeiten?
	Kategorie: Aus- und Weiterbildung	Diese Kategorie ist im Kontext des durchgeführten Projektes nur für die KollegInnen der Fachabteilung relevant.
F3	Vor der Phase der Implementierung wurde ein initiales Training durch die KollegInnen des RAD CoE abgehalten – wie beurteilen Sie den Inhalt des Trainings sowie dessen Sinnhaftigkeit?	- positive/negative Aspekte? - fehlende Inhalte?
F4	Anschließend habe Sie den empfohlenen 12h Onlinekurs im Selbststudium absolviert. Wie beurteilen Sie dessen Inhalt sowie Sinnhaftigkeit in Bezug auf Ihre darauffolgende Aufgabe im Projektteam?	- positive/negative Aspekte? - gut genug vorbereitet auf Projekteinsatz? - neben Tagesgeschäft durchführbar?
	Kategorie: Projekt Umsetzung	
F5	Wenn Sie sich an die Phase der Entwicklung zurückerinnern. Welche Punkte sind Ihnen dabei positiv und welche negativ in Erinnerung geblieben?	- fehlende Hilfsmittel? - ausreichend Dokumentation? - technische Hindernisse?

		- Umsetzungsmethode? - Einflüsse von außen? - sonstige Hindernisse?
	Kategorie: Kommunikation	
F6	Wie würden Sie rückblickend die Kommunikation im Projektteam mittels der zur Verfügung gestellten Kanäle sowie der definierten Regeln beurteilen?	- Regeln hinterfragen - Kanäle hinterfragen - Kommunikationshindernisse?
	Kategorie: Betrieb	Diese Kategorie ist im Kontext des durchgeführten Projektes nur für die KollegInnen der Fachabteilung relevant.
F7	Seit dem Go-Live wird die Anwendung direkt in der Fachabteilung weiterentwickelt – welche Hindernisse gibt es dabei und welche negativen oder positiven Aspekte ergeben sich durch die Zusammenarbeit mit dem RAD CoE (Deployments, Fragen, …)	- Kommunikation RAD CoE / Fachabteilung? - Hindernisse bei Umsetzung? - fehlende Werkzeuge / Dokumentation?
	Kategorie: Sonstiges	
F8	Gibt es sonst noch etwas das Sie anmerken wollen?	

Tabelle 4: Interviewleitfaden (eigene Darstellung)

3.2.3 **Durchführung und Aufbereitung des Interviews**

Alle am Projekt beteiligten Personen wurden bereits vor Projektbeginn darüber informiert, dass im Zuge dieser Masterarbeit das betreffende Projekt als Fallstudie genutzt und nach Projektabschluss ein ExpertInneninterview durchgeführt wird. Um eine möglichst ausgewogene und repräsentative Sicht auf das Projekt, dessen Ablauf sowie die Leistungen des ANDRITZ RAD CoE zu erhalten, wurden jeweils zwei Personen aus dem Fachbereich und ein Kollege aus der IT-Abteilung als zu interviewende ExpertInnen ausgewählt.

Die Interviews der ExpertInnen aus der Fachabteilung (aus Deutschland und der Schweiz) wurde via MS Teams durchgeführt. Der Kollege aus dem ANDRITZ RAD CoE wurde persönlich interviewt. Vor Beginn des Interviews wurden die ExpertInnen darüber aufgeklärt, dass das Interview zum Zwecke einer späteren Transkription und Auswertung aufgenommen wird. Waren die ExpertInnen damit einverstanden, hat das eigentliche Interview auf Basis des in Kapitel 3.2.2 beschriebenen Leitfadens begonnen. Auf ein Aktivieren der Webcam, bei den beiden via MS-Teams durchgeführten Interviews, wurde auf Wunsch der beteiligten ExpertInnen verzichtet. Aufgenommen wurden die Interviews im Falle von MS-Teams direkt mit der integrierten Aufnahmefunktion des Tools, bei den persönlichen Interviews wurde die Aufnahmefunktion des Smartphones (Huawei P10) benutzt.

Im Anschluss an die Interviews wurde dieses transkribiert, es wurden dabei die folgenden einfachen Transkriptionsregeln angewandt:

- Die interviewende Person wird durch ein „I“ gekennzeichnet.
- Die befragte Person wird durch die Abkürzung „Exp“ für ExpertIn, gefolgt von einer Kennnummer (z.B.: „Exp1“) gekennzeichnet.
- Wörter im Dialekt werden ins Schriftdeutsche übersetzt, der gesprochene Satz bleibt dabei erhalten – z.B.: „ist“ statt „is“.
- Transkribiert wird wörtlich, nicht lautsprachlich.
- Transkribiert wird zusammenfassend – es werden sich wiederholende oder abgebrochene Wörter und Sätze ausgelassen, sofern diese nicht relevant für das Interview sind.
- Wörter mit Betonung während des Interviews werden unterstrichen dargestellt, z.B.: „Wir mussten das so machen“.
- Alle Angaben, welche auf eine befragte Person rückschließen lassen, werden anonymisiert.
- Außersprachliche Ereignisse wie Lachen oder eine längere Pause werden in Klammer dargestellt, z.B.: [längere Pause], [lacht].

3.2.4 **Auswertung der Interviews**

Die aufgezeichneten Interviews wurden in einem ersten Schritt anhand der im vorigen Kapitel beschriebenen Regeln transkribiert und somit für die weitere Analyse vorbereitet. Als nächstes wurden die Daten strukturiert und darauf aufbauend analysiert – dazu stehen eine Vielzahl an Methoden zur Verfügung.

In dieser Masterarbeit wurde eine qualitative Inhaltsanalyse nach Mayring durchgeführt. Mit seiner Methode werden nicht nur die Texte analysiert, sondern diese Methode stellt darüber hinaus auch ein systematisches, regel- und theoriegeleitetes Vorgehen sicher (Mayring, 2002), welches den in Kapitel 0 definierten Gütekriterien entspricht.

Je nach Forschungsinteresse, können nach Mayring (2010) verschiedene Arten der Inhaltsanalyse angewandt werden:

- Zusammenfassende Inhaltsanalyse (Induktive Kategorienbildung): Das Datenmaterial wird auf die wesentlichen Aussagen reduziert und in Kategorien zusammengefasst. Die Bildung dieser Kategorien geschieht während der Inhaltsanalyse, aus dem Datenmaterial heraus.
- Strukturierende Inhaltsanalyse (Deduktive Kategorienbildung): Anhand von bereits vorab definierten, theoriebegründeten Kategorien, wird ein Querschnitt durch die Daten gelegt, welcher die Gesamtheit repräsentieren soll. Mayring (2010) sagt, die strukturierende Inhaltsanalyse hat das Ziel, bestimmte Aspekte aus dem vorhandenen Material herauszufiltern und somit das Material auf Grund bestimmter Kriterien einzuschätzen.

- Explikation: Wissenslücken werden durch das Hinzuziehen bzw. Einbringen von zusätzlichem Material geschlossen.

Im Mittelpunkt der qualitativen Inhaltsanalyse steht somit die Entwicklung eines Kategorie-Systems, welches das grundlegende Auswertungsinstrument darstellt und, mit Hilfe dessen, das verfügbare Material nach Strukturen und Kriterien durchsucht und analysiert wird. Im Zuge dieser Masterarbeit wurde eine strukturierende Inhaltsanalyse mit deduktiver Kategorienbildung durchgeführt, da, wie in Kapitel 4.1.1 ersichtlich, im Vorfeld der Fallstudie und der ExpertInneninterviews aus der wissenschaftlichen Literatur zentrale Einflussfaktoren identifiziert, und auf Basis derer ein Vorgehensmodell entwickelt wurde.

Das Kategoriesystem soll dabei laut Mayring (2010) so genau definiert werden, dass eine eindeutige und unmissverständliche Zuordnung von Textstellen möglich ist. Folgende Schritte werden dabei vorgeschlagen (Mayring, 2010):

1. Kategorien definieren: Genaue Definition, welche Textbestandteile einer Kategorie zugeordnet werden.
2. Definieren von Ankerbeispielen: Diese dienen als Beispiele für die zuvor definierte Kategorie – es werden konkrete Textstellen angeführt.
3. Kodierregeln erstellen: Dort, wo Abgrenzungsprobleme zwischen einzelnen Kategorien bestehen, werden Regeln formuliert, welche eine eindeutige Zuordnung ermöglichen (Mayring, 2010),

Der allgemeine Ablauf der qualitativen Inhaltsanalyse, ist abschließend zusammengefasst in Abbildung 18 ersichtlich:

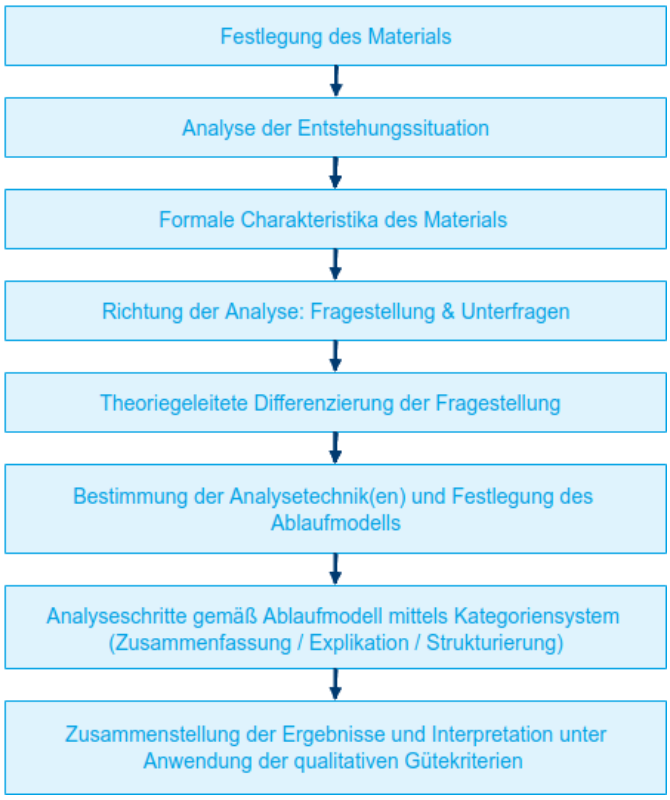

Abbildung 18: Ablaufmodell der qualitativen Inhaltsanalyse nach Mayring (eigene Darstellung in Anlehnung an Mayring, 2010)

3.2.5 **Gütekriterien**

Für die empirische Sozialforschung gilt, dass die gewonnen Ergebnisse des Forschungsprozesses anhand von Gütekriterien eingeschätzt werden sollen (Mayring, 2002). Solche Gütekriterien sind für die quantitative Forschung durch Objektivität, Reliabilität und Validität gegeben – diese lassen sich jedoch nicht ohne weiteres für qualitative Verfahren übernehmen. Daher wurden durch Mayring (2002) sechs allgemein gültige Gütekriterien für die qualitativen Verfahren definiert:

1. Verfahrensdokumentation: Der Forschungsprozess soll für alle nachvollziehbar sein – dazu ist es notwendig, die Zusammenstellung der Analysemethoden (Kapitel 3.1 und Kapitel 3.2) sowie die Durchführung und Auswertung (Kapitel 3.2.3, 3.2.4, 4.3 und 4.4) der Datenerhebung zu dokumentieren.
2. Argumentative Interpretationsabsicherung: Interpretationen sind im Bereich der qualitativen Sozialforschung von erheblicher Bedeutung, daher ist es notwendig diese argumentativ zu begründen, um daraus Rückschlüsse im Denken zu ermöglichen.
3. Regelgeleitetheit: Bestimmte Verfahrensregeln müssen eingehalten werden – beispielsweise mit Hilfe von Ablaufmodellen das Datenmaterial systematisch bearbeiten (siehe Kapitel 3.2.4)

4. Nähe zum Gegenstand: So nah wie möglich an die Alltagswelt der beforschten Subjekte anknüpfen – ein offenes und zugleich gleichberechtigtes Verhältnis mit den Betroffenen etablieren.
5. Kommunikative Validierung: Die Forschungsergebnisse können zusammen mit der beforschten Person auf ihre Gültigkeit überprüft werden, indem diese vorgelegt und diskutiert werden.
6. Triangulation: Indem mehrere Analysegänge verknüpft werden, versucht man die Qualität der Forschung zu vergrößern.

3.2.6 **Die ExpertInnen**

Folgende Tabelle gibt eine Übersicht der Personen, welche am ExpertInneninterview teilgenommen haben. Auf ein namentliches Nennen der Personen wird auf Wunsch der Teilnehmer verzichtet. Stattdessen wird jeder teilnehmenden Person eine ID (z.B.: „Exp1“ für ExpertIn 1) zugewiesen.

Spezielles Interesse bestand darin, die Sichtweise der KollegInnen des Fachbereichs zu erhalten, da die Kollegen des RAD Services die zur Verfügung stehenden Werkzeuge, Methoden und Kommunikationsmittel bereits in anderen Projekten erfolgreich eingesetzt haben. Daher wurde auf seitens des RAD-Services nur der Product Owner interviewt, die beteiligten EntwicklerInnen jedoch nicht.

ID	Rolle im Projekt (Bereich)	Biografische Information
Exp1	Business Developer (FA)	Männlich, 42 Jahre, 21 Jahre bei ANDRITZ beschäftig
Exp2	Citizen Developer (FA)	Männlich, 28 Jahre, 2 Jahre bei ANDRITZ beschäftigt
Exp3	Product Owner (IT)	Männlich, 36 Jahre, 8 Jahre bei ANDRITZ beschäftigt

Tabelle 5: Informationen zu den ExpertInnen, welche am Interview teilnahmen (eigene Darstellung)

Die Ergebnisse der ExpertInneninterviews sind in Kapitel 4.4 ersichtlich.

4 ERGEBNISSE

Dieses Kapitel beschreibt, das im Zuge dieser Arbeit entworfene Vorgehensmodell, welches innerhalb der ANDRITZ AG, zur Etablierung der modellgetriebenen Softwareentwicklung mittels Mendix, zum Einsatz kommt. Dieses Vorgehensmodell wurde wie in Kapitel 3.1.2 und Kapitel 4.3 beschrieben, einem Praxistest unterzogen – die Erkenntnisse aus diesem Praxistest und den im Anschluss geführten ExpertInneninterviews (Kapitel 3.2 und Kapitel 4.4) sind in der folgenden Beschreibung nicht berücksichtig, da diese nach der Entwicklung des Vorgehensmodells durchgeführt wurden.

4.1 Entwicklung des Vorgehensmodells

Wie bereits in Kapitel 2.2 beschrieben und in Abbildung 7 ersichtlich, besteht ein Vorgehensmodell aus verschiedenen Komponenten, an denen sich das in dieser Mastarbeit entwickelte Vorgehensmodell grob orientiert. Die einzelnen Punkte des Vorgehensmodells werden in folgende Tätigkeitsfelder unterteilt:

1. Projektmanagement (PM): Dieser Punkt beinhaltet sämtliche Faktoren, welche zur Steuerung bzw. Überwachung eines Projektes und dessen erfolgreichen Abschlusses notwendig sind.
2. Konfigurationsmanagement (KM): Alle nötigen Faktoren, um die erstellte Anwendung konfigurieren zu können. Das beinhaltet beispielsweise das Erstellen, Betreiben und Warten der Cloud Umgebung. Die eigentliche Entwicklung ist nicht Teil des KM.
3. Qualitätsmanagement (QM): Maßnahmen, welche sicherstellen, dass qualitativ hochwertige Software ausgeliefert wird, werden diesem Punkt zugeordnet.
4. Systementwicklung (SE): Sämtliche Faktoren, welche die Entwicklung (Modellierung) der Applikation betreffen bzw. beeinflussen oder dafür notwendig sind.
5. Sonstige (SO): Alle Faktoren, welche nicht zu einem der Punkte von 1 bis 4 zugeordnet werden können.

Außerdem werden die in Abbildung 7 ersichtlichen Rollenmodelle sowie Werkzeuge beschrieben und den einzelnen Tätigkeitsfeldern zugeordnet.

4.1.1 **Einflussfaktoren**

Entlang des SDLC ergeben sich verschiedene Einflussfaktoren, welche in einem Softwareprojekt zu berücksichtigen sind. Diese Faktoren tragen unterschiedlich stark zum Erfolg bzw. Misserfolg eines Projektes bei. Die Grundlage für die Auswahl der unten angeführten Faktoren war eine Literaturrecherche, auszugsweise wurden folgende Quellen verwendet, um die Liste zu erstellen: (Moll, et al., 2004), (Tsun & Dac-Buu, 2008) oder (Hairul Nizam Nasir & Sahibuddin, 2011). Zu berücksichtigen ist, dass diese Faktoren mit speziellem Blick auf die Bedürfnisse des ANDRITZ

RAD[11] Services sowie der Entwicklung mit Low-Code Plattformen ausgewählt wurden. Das bedeutet, das Faktoren wie beispielsweise Verträge, welche in Softwareprojekten mit externen Kunden eine bedeutende Rolle spielen können, nicht berücksichtigt wurden. Da die Plattform und die Infrastruktur vorgegeben sind, wurden diese Faktoren ebenfalls nicht miteinbezogen. Aus diesen identifizierten Faktoren wird im nächsten Schritt ein für das ANDRITZ RAD Service angepasstes Vorgehensmodell, bestehend aus Rollen, Prozessen, Best Practices und Regeln abgeleitet.

Die identifizierten Einflussfaktoren lauten wie folgt:

1. **Anforderungsanalyse:** Laut Jones (1997) sind falsch verstandene oder umgesetzte Anforderungen der häufigste Grund für das Scheitern von Software-Projekten. Anforderungen können von verschiedenen Projektbeteiligten unterschiedlich interpretiert werden, je später im Projektverlauf diese Missverständnisse aufgedeckt werden, desto teurer wird es, diese zu korrigieren (Bray, 2002). Es ist also unbedingt nötig, dass der Fachbereich und der Umsetzungspartner von anfang an eine gemeinsame Sprache sprechen und grundlegendes Domänen- bzw. Datenwissen seites des gesamten Projektteams vorhanden ist.
2. **Umsetzungsmethoden[12]:** Die gewählte Umsetzungsmethode wie beispielsweise RAD, Scrum, Kanban usw. kann sich von Projekt zu Projekt unterscheiden. Faktoren, welche die Wahl der passenden Umsetzungsmethode beeinflussen, sind unter anderem die Komplexität des Projektes, die zur Verfügung stehenden ProjektmitarbeiterInnen sowie die geplante Durchlaufzeit. Es ist auch möglich, nur Teile einer Umsetzungsmethode für ein bestimmtes Projekt heranzuziehen oder eine Mischform aus verschiedenen Methoden zu wählen. Die „richtige“ Umsetzungsmethode ist das Fundament des folgenden Projektes.
3. **Wissen:** Dieser Punkt beinhaltet das gesamte Know-How, welches das Projekt betrifft – von der Struktur der zugrunde liegenden Daten, den eingesetzten Werkzeugen, angebundenen Fremdsystemen, Konfiguration der Umgebungen (Cloud, on-Premises, …) und der gleichen. Insbesondere in einem dezentralisierten Team sind der Wissenstransfer bzw. das Verfügbar machen von Know-How ein essenzieller Bestandteil des Projekterfolges. Dieser Punkt ist eng verknüpft mit Punkt 8 – Dokumentation.
4. **Standards & Best Practices:** Gerade mit Blick auf wenig erfahrenen EntwicklerInnen aus den Fachbereichen ist es notwendig, Informationen zur Verfügung zu stellen, welche einerseits ein Set an grundlegenden Standards bzw. Regeln definieren, andererseits Hilfestellungen bezüglich der zu erledigenden Aufgaben bieten. Solche Hilfestellungen oder Best Practices sind beispielsweise bewährte Modellierungstechniken, welche in

[11] Innerhalb der ANDRITZ IT wurde der Service Rapid Application Development (RAD) gegründet, welcher die Nutzung, den Betrieb sowie die Weiterentwicklung bzw. Weiterverbreitung der Mendix Plattform innerhalb der ANDRITZ Gruppe zum Ziel hat.

[12] Normalerweise Vorgehensmodell genannt – zur besseren Unterscheidung zwischen dem in dieser Arbeit erstellten Vorgehensmodell und existierenden Vorgehensmodellen wird dieser Punkt Umsetzungsmethoden genannt.

jedem Projekt verwendet werden sollten, um die Wartbarkeit und Weiterentwicklung zu gewährleisten. Standards definieren, beispielsweise welche Voraussetzungen gegeben sein müssen, damit ein Arbeitspaket als abgeschlossen erachtet werden kann oder welche Dokumentationsmaßnahmen angemessen sind.

5. **Automatisierung:** Wie bereits in Kapitel 2.4 erwähnt, bringt die Automatisierung von sich wiederholenden Abläufen einige Vorteile mit sich. Beispielsweise wenn damit MitarbeiterInnen aus dem IT-Team entlastet werden können, da sie sich nicht auf das Überprüfen, Zusammenführen und Verfügbar machen von Codeänderungen der MitarbeiterInnen aus den Fachbereich kümmern müssen. Eine CI / CD Pipeline ist für den Projekterfolg zwar nicht unbedingt nötigt, gibt jedoch Ressourcen frei, welche anderwärtig genutzt werden können.
6. **Testen:** Obwohl Testen bereits im Punkt 5 (Automatisierung) als Bestandteil der CI / CD Pipeline enthalten ist, kann deshalb nicht auf manuelles Testen der Anwendung verzichtet werden. Typischerweise werden bei automatisierten Tests folgende Testarten durchgeführt: Modul- bzw. Komponententests, Schnittstellen- bzw. Integrationstests sowie GUI Tests. Diese Testarten stellen zwar die Funktionsfähigkeit und Fehlerfreiheit der Software nach einer Änderung sicher, jedoch ersetzten sie nicht die erfahrungsbasierten bzw. explorativen Tests, welche nur durch Menschen stattfinden können. Ebenso wenig können User Experience Tests automatisiert durchgeführt werden. Außerdem muss beim Erstellen einer CI / CD Pipeline auf das Kosten / Nutzen Verhältnis geachtet werden – für ein kleines und einfaches Projekt, bei dem wenige Codeänderungen durchgeführt werden, steht der Konfigurationsaufwand einer CI / CD Pipeline in keinem sinnvollen Verhältnis zur eingesparten Zeit. In solch einem Fall wird komplett auf Automatisierung verzichtet und die Anwendung manuell getestet.
7. **Feedback:** Regelmäßige und schnelle Rückmeldung vom Fachbereich bzw. Auftraggeber ist essenziell, um sicherzustellen, dass die Entwicklung in die gewünschte Richtung geht. Durch die direkte Integration des Fachbereichs in den Entwicklungsprozess und der daraus resultierenden engen Zusammenarbeit mit der Entwicklung sowie Operations, welche in Kapitel 2.4.1 beschrieben wird, können nicht nur Missverständnisse frühzeitig beseitigt werden, sondern durch die ständigen Rückmeldungen können auch Maßnahmen zur Prozessverbesserung abgeleitet und im Projekt umgesetzt werden.
8. **Dokumentation:** Das in Punkt 3 (Wissen) besprochene Know-How muss dokumentiert und an zentraler, für alle Teammitglieder zugänglicher Stelle, abgelegt werden. Diese Dokumentation dient einerseits als Basis für die Entwicklung (beispielsweise durch User-Stories), andererseits ist es notwendig, sicherzustellen, dass getroffene Entscheidungen auch nach längerer Zeit nachvollziehbar bleiben. Außerdem erleichtert ein gut dokumentiertes Projekt die Einarbeitung neuer KollegInnen. Das betrifft ebenso die Dokumentation der in Mendix erstellten Microflows – je nach Komplexität und Größe eines Microflows, kann das „Hineindenken“ in die zugrundeliegende Logik trotz visueller Komponenten schwierig werden, – besonders wenn ein Microflow andere Microflows aufruft usw.

9. **Kommunikation:** Durch die dezentrale Teamsituation ist effektive Kommunikation ein entscheidender Faktor. Dabei steht nicht nur der Austausch von Wissen oder die Koordination des weiteren Vorgehens während Besprechungen mit Mittelpunkt, sondern auch das einfache und ungezwungene in Kontakt treten mit den restlichen Mitgliedern des Teams, um ein Gemeinschaftsgefühl, Vertrauen und Respekt herzustellen. Wie bereits in Kapitel 2.5.1 erwähnt, gibt es in verteilten Teams gewisse Herausforderungen, welche es durch optimierte Kommunikation zu lösen gilt.
10. **Team:** Die Zusammensetzung sowie die Größe eines Projektteams sind kritische Faktoren für den Erfolg bzw. Misserfolg eines Softwareprojektes. Erneut mit Blick auf die EntwicklerInnen des Fachbereichs, welche wenig Erfahrung im Bereich der Softwareentwicklung mitbringen, ist es umso wichtiger, die nötige Unterstützung entweder in Form von erfahrenen EntwicklerInnen direkt im Team oder AnsprechpartnerInnen in Form von ExpertInnen, welche nicht im Team aber trotzdem greifbar sind, anzubieten. Die daraus resultierenden Rollen und Verantwortlichkeiten sind ein wichtiger Bestandteil. Ebenso fällt unter diesen Punkt die Möglichkeit zur Weiterentwicklung und den daraus resultierenden Rollenwechseln.

Um die identifizierten Einflussfaktoren je nach Projekt individuell beurteilen und auf deren Basis schnell und standardisiert die nötigen Entscheidungen treffen zu können, wurde im Zuge dieser Masterarbeit ein Center of Excellence (CoE) entworfen und innerhalb des ANDRITZ RAD Services gegründet. In diesem CoE fließen alle Informationen vor Projektbeginn sowie während der Projektumsetzung zusammen, um je nach Projekt- bzw. Ressourcensituation angemessene Entscheidungen treffen zu können. Der Aufbau sowie die genauen Verantwortlichkeiten des CoE werden in den folgenden Kapiteln näher erläutert.

4.2 ANDRITZ RAD - Center of Excellence (CoE)

Das CoE gibt die Vision des ANRITZ RAD Services vor - ausgerichtet an der IT-Strategie der globalen ANDRITZ IT. Es definiert Regeln, Prozesse und Best Practices, welche von den Teams genutzt werden und treibt die kontinuierliche Verbesserung voran. Außerdem legt es Rollen und Verantwortlichkeiten, die zur Verfügung stehenden Werkzeuge sowie das Aus- & Weiterbildungskonzept fest. Eine Übersicht der Verantwortlichkeiten des ANDRITZ RAD CoE ist in Abbildung 19 ersichtlich:

Abbildung 19: ANDRITZ RAD Center of Excellence (eigene Darstellung)

Das CoE soll, neben der Steuerung- und Führungsfunktion, außerdem als zentrale Anlaufstelle für sämtliche Anliegen, welche während des RAD Entwicklungsprozesses auftreten können, dienen. Das können einfache Fragen von EntwicklerInnen aus der Fachabteilung sein oder die Koordination und Abwicklung von Supportmaßnahmen mit externen Partnern wie SAP oder Mendix. Fixe Bestandteile dieses CoE sind der Plattform Owner, der Development Owner, ein Enterprise Architekt, ein Solution Architect, der App Delivery Manager sowie ExpertInnen für UI / UX und Softwareentwicklung, welche bei Bedarf in verschiedenen Projekten hinzugezogen bzw. konsultiert werden können.

Die Vision des ANDRITZ RAD Services lautet:

> *We are the first-choice partner for tailored and streamlined cloud based software solutions to increase the competitive advantage of ANDRITZ.*

Die einzelnen Bereiche des CoE sowie deren Aufgaben, werden in den folgenden Kapiteln näher beschrieben. Die einzelnen Teilbereiche bzw. Werkzeuge werden jeweils einem Tätigkeitsfeld aus Kapitel 4.1 zugeordnet.

4.2.1 **Methoden**

Unter diesen Teilbereich des CoE fallen sämtliche Werkzeuge, welche innerhalb des ANDRITZ RAD Services genutzt werden. Außerdem wird das Vorgehen bezogen auf unterschiedliche Projektphasen definiert. Der erste Kontaktpunkt eines Fachbereiches mit dem CoE ist meist der Bedarf nach einer neuen Anwendung. Wie mit solchen Anforderungen umgegangen werden soll und welche weiteren Maßnahmen darauffolgen, wird in den folgenden Kapiteln beschrieben.

4.2.1.1. Anforderungsmanagement und Projekt Setup (PM)

Anforderungen, welche entweder direkt vom Fachbereichen im CoE eingelastet, oder durch einen Business Relationship Manager (BRM) vom Fachbereich entgegengenommen und and das CoE weitergegeben werden, durchlaufen einen einfachen Prozess, an dessen Ende entweder die Entwicklung im RAD Umfeld mittels Mendix stattfinden kann oder weiterführende Maßnahmen ergriffen werden müssen. Diese weiterführenden Maßnahmen sind beispielsweise der Verweis auf bereits bestehende Anwendungen, mit ähnlicher / identer Funktionalität oder das Weitergeben der Anforderungen an einen IT-Bereich, welcher die Anforderung besser bedienen kann. Die Beurteilung, ob die Anforderung in das RAD Portfolio passt, obliegt dem ANDRITZ RAD Enterprise Architekten (siehe Kapitel 4.2.7).

Um die Anforderungen kategorisieren und darauf aufbauend weiterführende Entscheidungen - beispielsweise die Zusammensetzung des Teams oder die zu verwendende Umsetzungsmethode - treffen zu können, wurde eine Matrix entwickelt. Diese Matrix zieht einerseits die zu erwartende Komplexität der Umsetzung in Betracht andererseits auch ob die Anwendung lediglich ANDRITZ intern zum Einsatz kommt oder ob externe Kunden damit interagieren werden. Anforderungen, welche als „Low Complexity & Low Exposure“ eingestuft werden, erfordern beispielsweise nicht notwendigerweise den Einsatz eines Solution Architekten. Solche Projekte eignen sich für die Umsetzung in der Mendix No-Code Umgebung und können großteils selbstständig von der Fachabteilung durchgeführt werden. Die IT-Abteilung stellt dabei mittels dem CoE lediglich eine Anlaufstelle bei Entwicklungsproblemen sowie die Konfiguration der Cloud Umgebung zur Verfügung. Die erwähnte Matrix ist in Abbildung 20 ersichtlich:

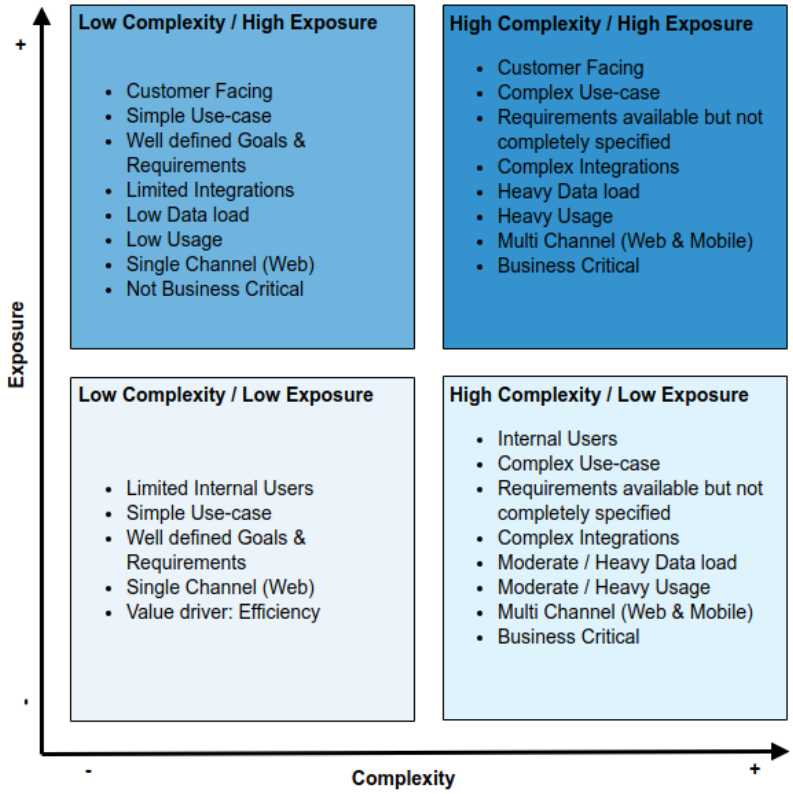

Abbildung 20: Kategorisieren der Anforderungen innerhalb des ANDRITZ RAD CoE (eigene Darstellung angelehnt an Mendix 2020)

Anhand der Kategorisierung der Anforderung in eine der vier Kategorien, können Maßnahmen betreffend der notwendigen Rollen innerhalb des Teams, der zu verwendenden Werkzeuge oder dem erforderlichen Minimum an Dokumentation, abgeleitet werden. Komplexe Anwendungen, bei denen mit erhöhtem Datenaustausch mit anderen Systemen zu rechnen ist, erfordern beispielsweise den Einsatz eines Solution Architekten um Performanceprobleme so gut als möglich ausschließen zu können. Anwendungen, welche außerhalb des Unternehmens verfügbar sind, müssen darüber hinaus auch strengere Anforderungen an das UI und UX erfüllen, um bei externen Kunden einen bestmöglichen Eindruck zu hinterlassen.

Die nötigen Informationen, um die Beurteilung im CoE durchführen zu können, werden in einem Kick-Off Meeting eingeholt. Verpflichtende Teilnehmer dieses Meetings sind, neben Vertretern des Fachbereichs (wichtig sind hier beispielsweise Key User), zumindest ein erfahrener Kollege / eine erfahrende Kollegin des CoE (Enterprise Architect, Solution Architect, Core Developer), welcher die nötigen Anforderungen in einem standardisierten Confluence Template, welches angelehnt an die Software Requirements Specifiaction (SRS) des IEEE (IEEE Standard Association, 2020) die für ANDRITZ notwendigen Elemente dokumentiert. Diese Elemente sind auszugsweise das Ziel und der Zweck der Anwendung, eine allgemeine Beschreibung der zu entwickelnden Lösung, die Bedürfnisse der Anwender, welche damit abgedeckt werden sollen sowie spezifische, als kritisch erachtete, Systemfunktionen. Außerdem wird noch die Anzahl der erwarteten Nutzer oder möglichen Interaktionen mit anderen Systemen erfasst. In diesem Meeting wird auch die generelle Bereitschaft zur Mitentwicklung sowie die Entwicklungskompetenz der interessierten MitarbeiterInnen des Fachbereiches erfragt und beurteilt.

Abgeleitet aus der Anforderungskategorie, stehen prinzipiell drei Varianten der Zusammenarbeit zwischen Fachabteilung und IT-Abteilung zur Verfügung – diese sind in Abbildung 21 ersichtlich:

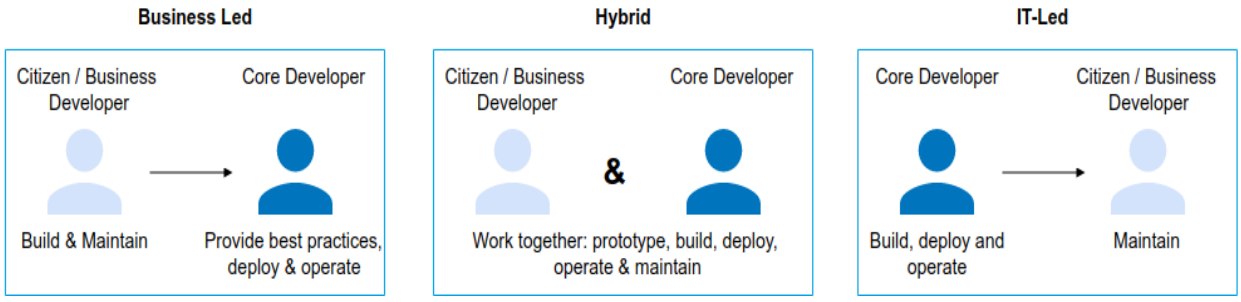

Abbildung 21: Modelle zur Zusammenarbeit zwischen IT und Fachbereich (eigene Darstellung in Anlehnung an Mendix, 2020)

Bei einfachen Anwendungen, welche in der Matrix links unten kategorisiert werden, liegt die Leitung, die Entwicklung sowie die Wartung direkt bei der Fachabteilung. Die IT ist lediglich für die Provisionierung, das Deployment sowie den Betrieb der Anwendung verantwortlich und steht für Fragen zur Verfügung. Dadurch, dass die notwendigen Schritte, um eine Anwendung in die Cloud Umgebung zu deployen ausschließlich von MitarbeiterInnen der IT durchgeführt werden

kann, ist sichergestellt, dass das Entstehen einer Schatten IT durch die Verwendung einer Plattform wie Mendix, siehe Kapitel 2.1.1.1, unterbunden wird.

Wird ein hybrider Ansatz der Zusammenarbeit gewählt, arbeiten die IT-Abteilung und der Fachbereich gemeinschaftlich und gleichberechtigt an einer Lösung. Wer dabei die Rolle des Product Owner übernimmt, wird einerseits aufgrund der Auslastungssituation der beteiligten Bereiche, andererseits aufgrund der Erfahrung der beteiligten ProjektmitarbeiterInnen entschieden. Es ist auch möglich, das Projekt mit zwei Projektleitern, jeweils aus der IT und dem Fachbereich, umzusetzen. Dies entspricht dem in Kapitel 2.5.2 beschrieben Faktor „geteiltes Führen“ für effektive Zusammenarbeit in verteilten Teams.

Liegt die Leitung, die Entwicklung sowie der Betrieb auf Seiten der IT-Abteilung, ähnelt die Vorgehensweise sehr stark einem klassischen Softwareprojekt. Der einzige Unterschied ist, dass einfache Änderungen nach dem Go-Live von der Fachabteilung selbst umgesetzt werden können und der Punkt Wartung somit in deren Verantwortung liegt. Bei Änderungen, welche die Kompetenz der Citizen bzw. Business Developer überschreiten, kann durch den CoE Expert Service (siehe nächstes Kapitel) Hilfestellung geboten bzw. bei zu komplexen Aufgaben die Umsetzung komplett übernommen werden.

Abhängig von der Komplexität des Projektes, kann es auch nötig sein, weiterführende Anforderungsanalysen bzw. Workshops abzuhalten, bei denen sich alle notwenigen Stakeholder sowie Vertreter des CoE und ggf. Vertreter anderer IT-Bereiche, deren Systeme ebenfalls betroffen sind, abstimmen. Die Problemstellungen, welche sich aus komplexen Anforderungen ergeben, werden mittels Design Thinking Workshops aufgelöst.

4.2.1.2. Design Thinking Workshops (SO)

Unter Design Thinking versteht man eine spezielle Herangehensweise zur Bearbeitung komplexer Probleme, welche Prinzipien die mit den Überschriften „Team“, „Prozess“ und „Raum“ beschrieben werden können, beinhaltet (Poguntke, 2020):

- Team: Die Teilnehmer eines Design Thinking Workshops sollten aus unterschiedlichen fachlichen Disziplinen stammen und ihr jeweiliges Fach- und interdisziplinäres Wissen lösungsorientiert einbringen.
- Prozess: Besteht aus den Phasen (Poguntke, 2020)
    - Problemkontext: Durch Beobachtung der derzeitigen Arbeitsweise oder Gesprächen bzw. Interviews mit AnwenderInnen (IST-Analyse) soll das Problem im Kontext der Arbeitsumgebung bzw. Arbeitsweise der NutzerInnen erfasst werden.
    - Definition: Der identifizierte Problemkontext wird konkretisiert und in Form eines Problemstatements festgehalten. Dabei steht die Zielgruppenorientierung im Mittelpunkt (SOLL-Entwicklung).

  - Ideenfindung: Erkenntnisse und Ideen zur Lösung des Problems werden in Form von Skizzen visualisiert, Abläufe können in Form von Customer Journeys dargestellt werden.
  - Prototyping: Mittels Mendix können die entwickelten Ideen schnell und mit geringem Aufwand in Prototypen umgewandelt und somit testbar gemacht werden.
  - Test: Durch mehrere Iterationen der Prototypen und dem Testen dieser, nähert man sich der idealen Lösung schrittweise an, bis diese schließlich gefunden wurde.
- Raum: Der Raum stellt genügend Platz und alle nötigen Hilfsmittel zur Verfügung, um das zu bearbeitende Problem kreativ zu lösen. Das beinhaltet beispielsweise Whiteboards, Haftnotizen, Moderationskoffer mit Materialen, um ggf. auch Papier-Prototypen herstellen zu können usw.

Sind die Anforderungen spezifiziert, die am Projekt beteiligten Personen identifiziert und festgehalten, unter welcher Leitung die Umsetzung erfolgen soll, muss sich vor dem eigentlichen Projektstart noch auf die passende Umsetzungsmethode geeinigt werden.

4.2.1.3. Umsetzungsmethoden (PM)

Wie bereits in Kapitel 2.2 erwähnt, ist die Wahl der passenden Umsetzungsmethode von mehreren Faktoren abhängig. Auch wenn sich RAD durch seinen prototypischen Ansatz als geeignetes Modell für Low-Code Entwicklungen erwiesen hat, bedeutet das nicht, dass sämtliche Projekte damit auf eine effiziente Art und Weise umsetzbar sind – ist das Projekt beispielsweise als klassisches Projekt (Leitung bei der IT) ausgelegt, kann nicht immer davon ausgegangen werden, dass sich der Fachbereich die enge und zeitintensive Zusammenarbeit, welche sich beim Einsatz von RAD durch die häufigen Iterationen von Prototypen und deren Tests ergeben, leisten will. Genauso wenig macht es Sinn, bei einem kleinen Projekt, welches unter Leitung der Fachabteilung stattfindet, Scrum als Methode, mit sämtlichen im Agilen Manifest beschriebenen Artefakten, wie Product- oder Sprint Backlog oder sämtlichen vorgeschlagenen Zeremonien, zu nutzen. Aus diesem Grund ist ein Festlegen auf eine Umsetzungsmethode für sämtliche Projekte innerhalb des RAD Services nicht sinnvoll. Oft kann es zielführender sein, mehrere Methoden zu kombinieren und die jeweils passenden Elemente aus den einzelnen Methoden individuell hinzuzufügen oder zu entfernen. Dieses Vorgehen ist als hybrides Projektmanagement bekannt.

Es stehen verschiedene Modelle zur Verfügung, welche die Wahl des „richtigen“ Umsetzungsmodelles erleichtern können.

Die grundlegende Entscheidung, ob als Ausgangspunkt eher eine traditionelle oder eine agile Methode angewandt werden soll, kann durch fünf von Boehm und Turner (2004) definierten Faktoren, welche in Abbildung 22 ersichtlich sind, getroffen werden:

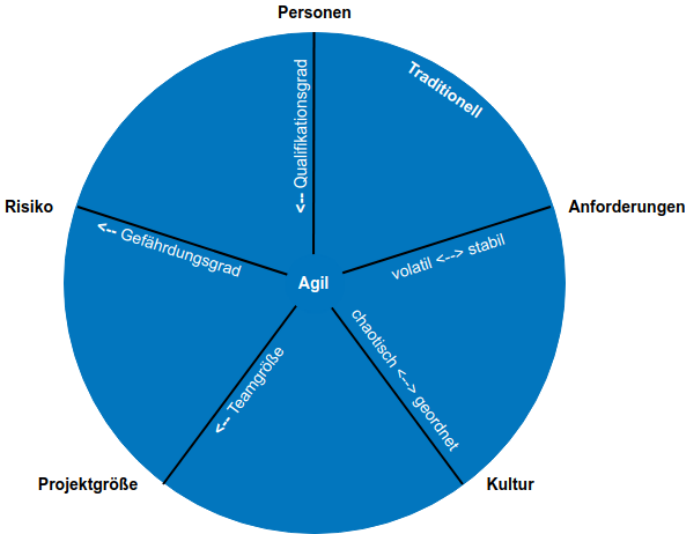

Abbildung 22: Auswahlmethode Vorgehensmodell 1 (eigene Darstellung in Anlehnung an Boehm & Turner, 2004)

Eine auf der Komplexität des Projektes sowie der Unsicherheit der Anforderung beruhende Methode, zur Auswahl einer geeigneten Umsetzungsmethode, wurde von Ebert (2014) entwickelt, welche in Abbildung 23 ersichtlich ist.

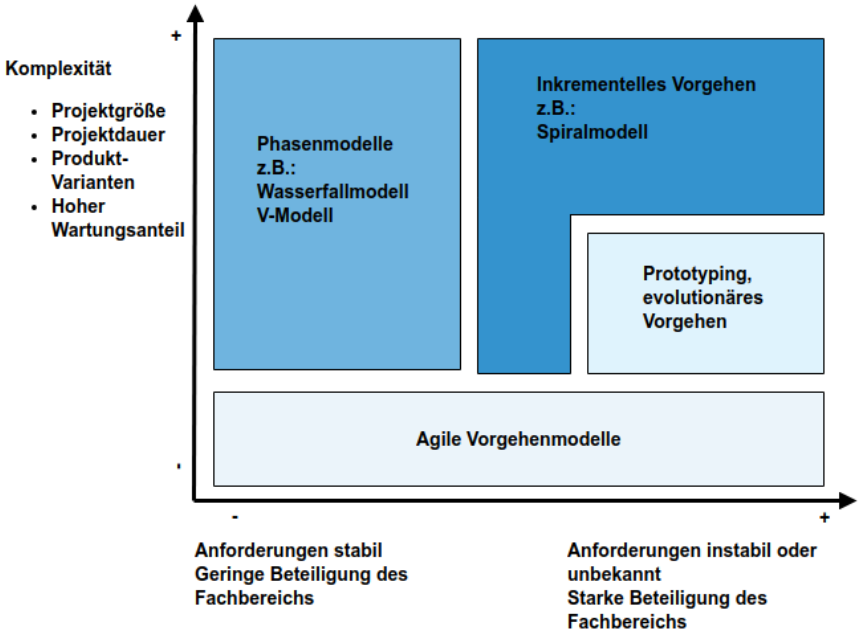

Abbildung 23: Auswahlmethode Vorgehensmodell 2 (eigene Darstellung in Anlehnung an Ebert, 2014)

Wie bereits erwähnt, dient das gewählte Umsetzungsmodell in manchen Fällen lediglich als Ausgangspunkt, welcher durch das Anreichern mit Elementen aus anderen Modellen oder durch das Weglassen von Elementen individuell angepasst werden kann.

4.2.1.4. Testen (QM)

Innerhalb des ANDRITZ RAD Services wird nach dem „whole-team approach“ gearbeitet, das bedeutet, dass jedes Teammitglied gleichermaßen für die Qualität und damit auch den Projekterfolg verantwortlich ist (Rouse, 2011). Dieses Vorgehen passt ebenfalls zum im Kapitel 2.4.1 beschriebenen BizDevOps Ansatz. Dadurch, dass jedes Teammitglied die Position eines Testers innehat, ergeben sich einige Vorteile. Beispielsweise lernt jedes Teammitglied die Anwendung als Ganzes besser kennen, da es „fremde Funktionen“ testen muss. Gleichzeitig wird die Kommunikation und Zusammenarbeit zwischen allen Teammitgliedern gefördert und verbessert. Der „whole-team approach“ kommt ursprünglich aus dem Bereich des agilen Testens.

Abhängig von der Komplexität sowie Sichtbarkeit (siehe Kapitel 4.2.1.1) der zu erstellenden Anwendungen wird entschieden, welche zusätzlichen Maßnahmen, wie beispielsweise das Nutzen von Automatisierung durch das Erstellen einer CI / CD Pipeline (siehe Kapitel 2.4), notwendig sind. Diese Entscheidung wird durch das Verwenden einer Risikomatrix unterstützt, welche den zu erwartenden Schaden bei Einritt einer Fehlfunkton bzw. dem Ausfall bestimmter Funktionen einer Anwendung, mit der Eintrittswahrscheinlichkeit eines solchen Vorfalls, gegenüberstellt. Das Ziel ist es, den Testaufwand je nach zu erwartendem Risiko der Anwendung, in einem wirtschaftlich vernünftigen Rahmen zu halten. Dies wird in Abbildung 24 dargestellt und wird als risikobasiertes Testen bezeichnet.

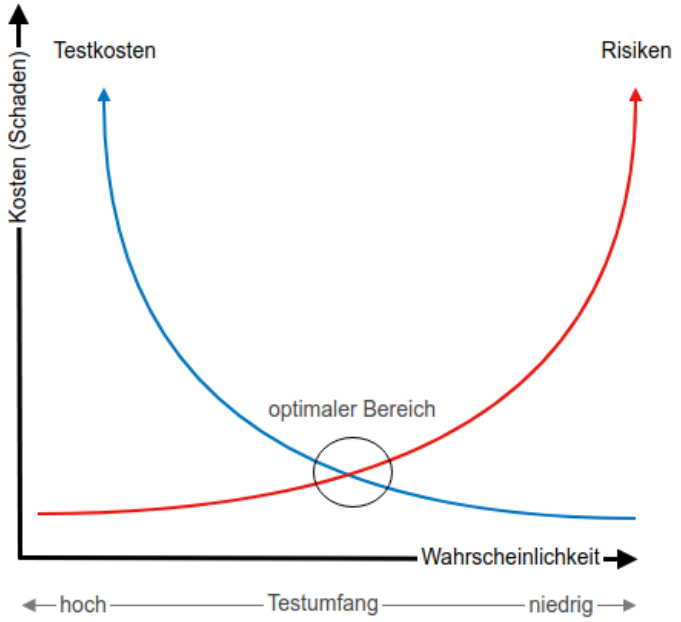

Abbildung 24: Risikobasiertes Testen (eigene Darstellung in Anlehnung an Arbeitskreis Testmanagement, 2004)

4.2.2 **Werkzeuge**

Je nach Rolle (siehe Kapitel 4.2.7) im Entwicklungsteam, stehen verschiedene Werkezeuge zur Verfügung, um die Anforderungen, welche aus der Rolle resultieren, zu erfüllen. Ganz nach dem in Kapitel 2.4 beschriebenen Lean Ansatz (Vermeidung von Verschwendung, Reduktion auf das Wesentliche) wurden die verfügbaren Werkzeuge auf ein Minimum reduziert. Tabelle 6 gibt auszugsweise eine Übersicht der im RAD Service zur Verfügung stehenden Werkezuge und teilt diese eine der in Kapitel 4.1. beschriebenen Kategorien des Vorgehensmodells zu.

Werkzeug	Zielgruppe	Einsatzzweck	Kategorie
Mx Studio	EntwicklerInnen	• No-Code Entwicklungsumgebung	SE
Mx Studio Pro	EntwicklerInnen	• Low-Code Umgebung	SE
IDE (Eclipse, IntelliJ)	ExpertInnen	• Erweiterung der Mendix Plattform	SE
Cloud Foundry CLI / Web	EntwicklerInnen / Operations	• Provisionierung[13] und Konfiguration der Cloud Umgebung	KM
Moqups	Alle	• Entwerfen UI / UX • Demonstration UI / UX	SO
Application Performance Monitor (APM)	EntwicklerInnen / Operations	• Load Tests • Monitoring der Verfügbarkeit/ Performance	QM / KM
Application Test Suite (ATS)	EntwicklerInnen	• Funktionale Tests • Regressionstests • Performance Tests • Continuous Integration	QM / KM
Jira	Alle	• Aufgabenmanagement • Fehlermanagement	PM
Confluence	Alle	• Standards • Best Practices • Dokumentation • Wissensdatenbank • FAQ	PM / QM / SO

[13] Unter Provisionierung versteht man das Bereitstellen von Ressourcen in einer Cloud Umgebung sowie das zuweisen dieser zu einem Projekt

MS Teams	Alle	• Enge Kollaboration innerhalb eines Projektteams • Zentrale Informationsquelle • Wissensaustausch	PM / QM / SO
Yammer	Alle	• Community • Socializing	SO
Matomo	Dev/ Ops	• Webanalytik	KM / SO
Jenkins	Dev / Ops	• CI / CD Automatisierung	KM / QM

Tabelle 6: ANDRITZ RAD Werkzeuge (eigene Darstellung)

4.2.2.1. Kommunikation (PM, QM, SO)

Da es sich bei den Mitgliedern des ANDRITZ RAD Services großteiles um MitarbeiterInnen handelt, welche nicht am selben Ort angesiedelt sind, kommt der effektiven Kommunikation und Zusammenarbeit innerhalb der verteilten Teams besonderen Bedeutung zu. Die Schwierigkeiten, welche durch dezentrale Teams entstehen können, wurden bereits in Kapitel 2.5 beschrieben. Innerhalb des ANDRITZ RAD Services wird als zentrales Kommunikations- und Kollaborationsmedium Microsoft Teams eingesetzt. Es wird dabei für jedes Projekt ein eigenes Team[14] erstellt, welches alle relevanten Personen vernetzt und in dem jegliche Informationen, welche das Projekt betreffen, verwaltet und gesammelt werden. ProjektmitarbeiterInnen müssen somit nur ein einziges Werkzeug nutzen, um einerseits mit anderen Teammitgliedern in Kontakt zu treten (sei es durch Videokonferenzen, Postings oder Chats), andererseits um Zugang zu sämtlichen projektrelevanten Dokumenten zu erhalten. Obwohl beispielsweise User Stories, FAQ-Artikel oder Besprechungsnotizen in Confluence erstellt und verwaltet werden sowie das Aufgabenmanagement in Jira organisiert wird, ist durch die Erweiterbarkeit von MS-Teams (durch das Einbinden von Werkzeugen) gewährleistet, dass auch solche Informationen innerhalb des Teams und somit zentral abrufbar sind. Der Fokus von Microsoft Teams liegt dabei auf der effektiven Zusammenarbeit sowie dem Wissensaustausch innerhalb eines Projektteams und dienst sozusagen als digitales Büro. Sämtliche Besprechungen werden über dieses Werkzeug abgehalten und jegliche Kommunikation das Projekt betreffend muss über diesen Kanal erfolgen. Dies wurde auch als Standard (siehe Kapitel 4.2.5) definiert.

Um diese Zusammenarbeit sowie den Wissensaustausch auch über einzelne Projektteams hinaus zu ermöglichen, ist neben Microsoft Teams noch Yammer (ebenfalls von Microsoft) innerhalb des RAD Services im Einsatz, welches als soziales Unternehmensnetzwerk betrachtet werden kann. Das Ziel, welches durch das Verwenden von Yammer erreicht werden soll, ist eine RAD Community innerhalb der ANDRITZ Gruppe zu schaffen. Einerseits sollen sich dadurch die global verteilten und aus unterschiedlichen Fachbereichen stammenden Mitglieder besser

[14] Als Teams werden im MS-Teams Kontext Sammlungen von Personen, Inhalten und Werkzeugen für unterschiedliche Projekte innerhalb einer Organisation bezeichnet (Microsoft Docs, 2020)

kennenlernen - es stehen beispielsweise Gruppen zur Verfügung in denen man sich abseits des Projektgeschäfts zwanglos austauschen kann -, andererseits soll durch Gruppen, in denen sich die Mitglieder gegenseitig bei Problemen betreffend der Entwicklung mit Mendix helfen können, ein reger Erfahrungsaustausch stattfinden, welcher in weiterer Folge auch das Zusammengehörigkeitsgefühl stärken kann. Des Weiteren wird Yammer auch genutzt, um Informationen zu verbreiten, welche für Personen im Umfeld des RAD Services relevant sind. Beispielsweise Informationen bezüglich Versionsupdates oder die Info über einen gelungenen Projektabschluss, um diesen gemeinsam feiern zu können. Außerdem werden virtuelle Team Building Möglichkeiten vorgestellt, wie beispielsweise der Einsatz von „QuizBreaker“ (Quizbreaker, 2020). Einzelne Projektteams sind angehalten, solche virtuellen Events abzuhalten, um das Team näher zusammenrücken zu lassen und das Teamwork zu verbessern.

Im Punkt Kommunikation ist auch definiert, wie im Falle eines auftretenden Problems während der Entwicklung mit Mendix vorgegangen werden soll. Erste Anlaufstelle, um eine mögliche Lösung für ein Problem zu finden, sind die im Confluence erstellten FAQ Artikel, welche häufig auftretenden Probleme sammeln und Lösungswege beschreiben. Sollte das Problem dadurch nicht lösbar sein, ist der nächste Schritt, das Problem im Team zu besprechen und in Yammer zu veröffentlichen, um innerhalb der ANDRITZ RAD Community eine Lösung zu finden. Ist auch dieser Weg erfolglos, wird ein im CoE angesiedelter Experte versuchen das Problem zu lösen. Das Anfordern eines solchen Experten / einer solchen Expertin erfolgt durch das Erstellen eines JIRA Tickets. Das Konsultieren von externen ExpertInnen, beispielsweise der Support von Mendix oder SAP, erfolgt je nach Bedarf, wird CoE intern veranlasst und durch den Plattform Owner initiiert.

4.2.3 **Plattform Management (KM, SO)**

Unter dem Punkt Plattform Management werden sämtliche Aufgaben zusammengefasst, welche mit dem Betrieb, der Wartung und Aktualisierung sowie dem Support, der im RAD Service Umfeld verwendeten Werkezuge zu tun hat. Das beinhaltet beispielsweise auch die transparente Weiterverrechnung der anfallenden Betriebskosten einer Anwendung, welche in der SAP-Cloud gehostet wird, an den entsprechenden Fachbereich. Die Höhe der Kosten ergibt sich anhand der geschätzten Anzahl der Benutzer, der benötigten Ressourcen (wie die Größe der Datenbank, zugewiesenem RAM usw.) sowie der integrierten Schnittstellen zu anderen Systemen. Der errechnete Betrag wird jährlich neu evaluiert und den veränderten Bedingungen (beispielsweise deutlich mehr Nutzer, woraus auch ein erhöhter Bedarf an Datenbankspeicher resultiert) angepasst.

4.2.4 Expert Services & Wiederverwendbare Komponenten (KM, SE, SO)

Im CoE Bereich Expert Services werden unterschiedliche Leistungen angeboten, welche die Entwicklungsteams im RAD Umfeld in Anspruch nehmen können. Beispielsweise stehen erfahrene Mendix EntwicklerInnen oder UI und UX ExpertInnen zur Verfügung, die ein Team bei Bedarf unterstützen können. Bei komplexeren Projekten mit höherem Risiko, sind solche ExpertInnen zu einem gewissen Prozentsatz fixer Bestandteil des Projektteams. Bei einfacheren Projekten können über das CoE „Beratungstermine“ vereinbart werden, um spezifische Probleme zu lösen. Sollte ein Problem ANDRITZ intern nicht lösbar sein, werden externe ExpertInnen, wie in Kapitel 4.2.2.1 beschrieben, hinzugezogen. Verantwortlich für den Aufbau der Kommunikation mit allen externen Partnern ist der Plattform Owner.

Ein weiterer Fokus liegt auf wiederverwendbaren Komponenten. Bestimmte Funktionalitäten innerhalb einer Anwendung können für andere Anwendungsfälle relevant sein. In vielen Fällen gibt es dafür bereits ein Widget im Mendix App Store, jedoch nicht in jedem Fall. Ziel ist es daher, solche wiederverwendbaren Komponenten idealerweise bereits währen der Konzeptionsphase des Projektes oder auch während der Entwicklung zu identifizieren. Ein Expert Developer hat anschließend die Aufgabe, diese Funktionalität umzusetzen und als Widget im Andritz internen „Company App Store“ zur Verfügung zu stellen.

Der Einsatz von ExpertInnen in den jeweiligen Teams, je nach Komplexität sowie das Extrahieren von wiederverwendbaren Komponenten aus unterschiedlichen Anwendungen, ist in Abbildung 25 ersichtlich:

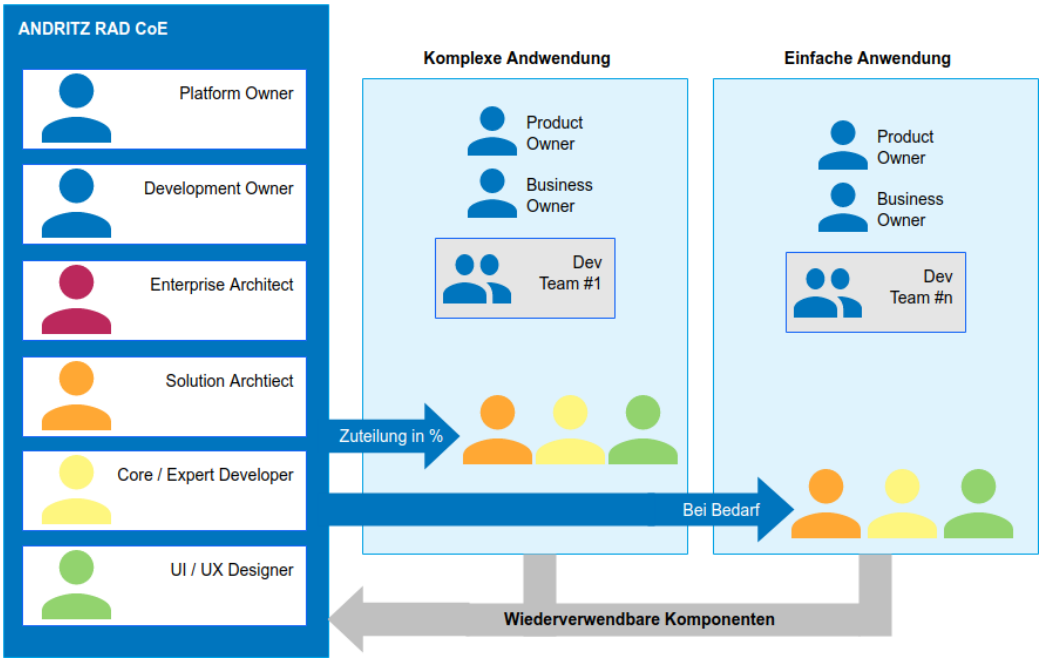

Abbildung 25: Expert Services des ANDRITZ CoE (eigene Darstellung)

4.2.5 **Standards & Best Practices (QM, SE, SO)**

Das Ziel der definierten Standards und Best Practices ist es, möglichst nachhaltige Anwendungen zu entwickeln, welche einfach wart- und erweiterbar sind. Außerdem soll sichergestellt werden, dass bereits während der Entwicklung die Dokumentation fester Bestandteil des Prozesses ist. Sollte der Fall eintreten, dass eine bereits entwickelte Anwendung auch in anderen Bereichen genutzt werden soll - beispielsweise erhält das CoE eine Anforderung, welche einer bestehenden Anwendung sehr ähnelt -, muss sichergestellt sein, dass auch an der Entwicklung unbeteiligte Personen diese im festgelegten Umfang nutzen und ihre Entwicklung fortsetzen können (Schlauch, Meinel, & Haupt, 2018). Der Unterschied zwischen einem Standard und einer Best Practice ist, dass ein Standard zwingend in einem Projekt umgesetzt werden muss. Im Gegensatz dazu gibt ein Best Practice eine Empfehlung ab und kann flexibler als ein Standard umgesetzt werden.

Da die komplette Liste der Standards und Best Practices den Rahmen dieser Arbeit sprengen würde, sind aus beiden Bereichen jeweils nur Beispiele angeführt. Definierte Standards sind unter anderem:

- Komplexere Microflows mit mehr als 10 Aktivitäten oder mehr als 2 Entscheidungen müssen eine Kurzbeschreibung der Funktionalität sowie der erwarteten Parameter und Rückgabewerte enthalten (Dokumentation direkt im Microflow).
- Projektabschluss Workshop: Am Ende eines Projektes müssen die Erkenntnisse, welche die einzelnen Teammitglieder während der Umsetzungsphase gesammelt haben, in einem Lessons Learned Workshop besprochen und dokumentiert werden.
- Commit Nachrichten müssen mit der entsprechenden JIRA Ticket Nummer versehen werden.
- Sämtliche, das Projekt betreffende Kommunikation, muss in dem dafür vorgesehenen MS Teams „Team“ stattfinden.
- Namenskonventionen: Anhand der Namen einzelner Bestandteile der Anwendung soll bereits erkannt werden, welche Funktion diese besitzen. Beispielsweise gibt es folgende Konvention für Microflows: *{Prefix}_{Entität}_{Funktion}* - der Prefix gibt den Auslöser des Microflows an, beispielsweise steht BCo für „Before Commit“ oder DS für „Data Source“ (Mendix Docs, 2020). Danach folgt, welche Entität bearbeitet wird und zum Schluss die Operation, welche darauf durchgeführt wird. Zum Beispiel BCo_Address_FormatPostalCode. Ähnliche Namenskonventionen gibt es für Seiten.
- Entwicklungstemplate: Vom CoE wird ein Projekttemplate zur Entwicklung mittels Mendix zur Verfügung gestellt, welches alle notwendigen Module und Abhängigkeiten beinhaltet, um sofort mit der Umsetzung der Anforderung beginnen zu können. Beispielsweise sind in diesem Template das ANDRITZ UI enthalten, die notwendigen Module zur Authentifizierung und Autorisierung mittels ANDRITZ Active Direcotry (AD), Skripte zum Erfassen der grundlegenden Metriken (beispielsweise Anzahl der Zugriffe) sowie Backup Skripte, um automatisiert Datenbanksicherungen erstellen zu können. Dieses Template

ist der Ausgangspunkt für jede neu zu erstellender Anwendung und minimiert den Konfigurationsauswand.

Beispiele für Best Practices:

- Innerhalb einer Entität sollten optimalerweise keine berechneten Attribute verwendet werden – dies kann zu Performance Problemen führen, da die Kalkulation bei jeder Verwendung des Objektes ausgeführt wird (Mendix Docs, 2020).
- Hat ein x-Path mehrere Bedingungen, sollte jede Bedingung einzeln (mittels []) definiert werden, anstatt diese mit dem Keyword *and* zu verbinden (Mendix Docs, 2020).
- Während online Besprechungen sollte die Webcam aktiviert sein, damit man seine Teammitglieder auch sehen kann.
- Komplexe Prozesse und kritische Geschäftslogik sollten Informationen im Level „Debug“ oder „Trace“ in den Logfiles zur Verfügung stellen, um bei Problemen ohne Anpassung des Loglevels schnell reagieren zu können.
- Definiton of Ready (DoR): Beschreibt, welche Bedingungen erfüllt werden müssen, damit beispielsweise ein Arbeitspaket, welches umgesetzt werden soll, für die Entwicklung bereit ist. Notwendige Bedingungen sind auszugsweise: die Akzeptanzkriterien sowie die nicht funktionalen Anforderungen sind definiert, Abhängigkeiten zu anderen User Stories bzw. Funktionen innerhalb der Anwendung sind identifiziert, Anhängigkeiten zu anderen Teams (beispielsweise durch das Anbinden eines Fremdsystems) sind identifiziert, anhand des Risikos bzw. der Komplexität der Funktion wurde ein angemessener Test- und Dokumentationsaufwand definiert.
- Definiton of Done (DoD); Beschreibt, welche Bedingungen erfüllt werden müssen, damit ein fertig implementiertes Arbeitspaket als abgeschlossen angesehen werden kann. Die notwendigen Bedingungen sind auszugsweise: die Akzeptanzkriterien sowie die nicht-funktionalen Anforderungen sind erfüllt, die umgesetzte Funktion ist entsprechend des Risikoprofils getestet, die Funktion wurde in einer Art modelliert, dass diese selbsterklärend ist oder sie wurde entsprechend dokumentiert, die Änderungen sind vom lokalen Repository in das zentrale Projekt Repository gepusht worden.

All dieses Standards und Best Practices sind im ANDRITZ RAD Confluence Bereich einsehbar, welcher für jedes Mitglied des ANDRITZ RAD Services zugänglich ist und im Zuge des „Onboardings“ neuer KollegInnen, welche in den Dev-Pool aufgenommen werden, durchgesprochen wird.

4.2.6 **Kontinuierliche Verbesserung (PM, QM, SE, SO)**

Wie bereits im letzten Kapitel erwähnt, sind Lessons Learned Workshops verpflichtend am Ende eines Projektes durchzuführen, um Informationen aus den einzelnen Projekten in den kontinuierlichen Verbesserungsprozess einfließen zu lassen. Solche Informationen sollen beispielsweise aufzeigen, in welchen Bereichen während der Entwicklung Engpässe entstehen, ob die Strukturen des CoE in angemessener Weise unterstützen können oder, ob die zur Verfügung stehenden Werkzeuge ausreichend sind. Zusammengefasst geht es um die Fragen

- Was hat gut funktioniert?
- Welche Probleme hat es in welchen Phasen gegeben?
- Wo gibt es Verbesserungspotential?

Da bereits vor Projektbeginn bekannt ist, dass ein solcher Workshop stattfinden wird, haben sämtliche Teammitglieder die Möglichkeit ihre Erfahrungen bereits während der Laufzeit des Projektes zu dokumentieren. Die gewonnenen Erkenntnisse werden in einem ersten Schritt innerhalb des CoE analysiert. Es soll die Frage beantwortet werden, warum es die aufgezeigten Probleme gegeben hat. Welche Rahmenbedingungen (Werkzeuge, Umsetzungsmethode, Kommunikationskanäle usw.) haben dazu beigetragen? Diese Erkenntnisse werden dokumentiert und zentral in Confluence zugänglich gemacht. Aus diesen Erkenntnissen werden innerhalb des CoE geeignete Maßnahmen (wie beispielsweise neue bzw. angepasste Standards oder Werkzeuge) abgeleitet, um in zukünftigen Projekten einen optimierten Ablauf bzw. angepasste Hilfsmittel zur Verfügung stellen zu können, welche die erkannten Fehler vermeiden sollen und somit das Risiko eines Projektmisserfolges reduzieren. Der kontinuierliche Verbesserungsprozess folgt somit dem Plan-Do-Check-Act (PDCA) Prinzip.

Natürlich können kritische Verbesserungen auch direkt, während der Umsetzungsphase eines Projektes, identifiziert und bei Bedarf sofort behoben werden. Je nach gewählter Umsetzungsmethode, stehen dafür unterschiedliche Werkezeuge zur Verfügung. Wird mittels Scrum gearbeitet, bietet sich beispielsweise die Retrospektive am Ende eines Sprints an. Somit können kritische Mängel wie zum Beispiel fehlendes Know-How bzw. eine „falsche“ Teamzusammensetzung zeitnah behoben werden.

4.2.7 **Rollen & Verantwortlichkeiten (PM, SO)**

Die in einem Softwareprojekt typischen Rollen und Verantwortlichkeiten, werden durch die Einführung von Low-Code Entwicklungsplattformen erweitert. Dadurch, dass die Fachabteilungen direkt in den Entwicklungsprozess miteinbezogen werden können, müssen Personen aus diesen Bereichen ebenfalls mitberücksichtigt werden. Es genügt beispielsweise nicht, die EntwicklerInnen je nach Erfahrung in Junior- bzw. SeniorentwicklerInnen zu kategorisieren - dies würde voraussetzen, dass zumindest eine grundlegende Programmierausbildung vorhanden ist. Abbildung 26 zeigt die verschiedenen Rollen, welche ein(e) EntwicklerIn innerhalb der ANDRITZ AG einnehmen kann.

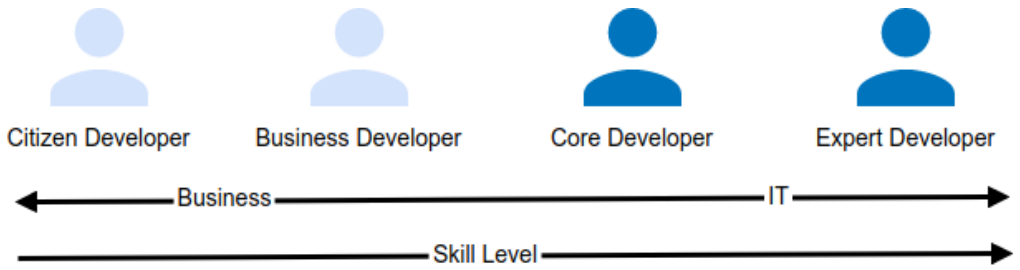

Abbildung 26: Low-Code Entwicklerrollen innerhalb des ANDRITZ RAD Services (eigene Darstellung, angelehnt an Mendix 2020)

- Citizen Developer: Typischerweise eine EndanwenderIn, welche direkt in der Fachabteilung sitzt und wenig bis keine Erfahrung im Bereich der Softwareentwicklung aufweist (Medix Evaluation Guide, 2019). Dieser Entwicklertyp besitzt grundlegendes technisches Verständnis und kann mit Hilfe von Mendix Studio einfache Anwendungen, welche Firmenintern genutzt und eine Steigerung der Produktivität zum Ziel haben, umsetzen.
- Business Developer: Fungiert als Bindeglied zwischen Citizen Developer und ExpertInnen der IT-Abteilung. Der Business Developer zeichnet sich durch Datenkompetenz aus, das bedeutet er kann mit den Daten einer bestimmen Domäne fachgerecht und wertgenerierend umgehen – er kann die Daten sammeln, anpassen, analysieren, fachgerecht bewerten, visualisieren und weiß, wie diese anzuwenden sind (Luber & Litzel, 2019). Er/Sie versteht die Anforderungen des Fachbereiches, hat das nötige Domänen- bzw. Prozesswissen sowie das technische Verständnis, um die Anforderungen umzusetzen (Medix Evaluation Guide, 2019). Business Developer werden auch „Subject Matter Expets“ (SME) genannt – diese Entwicklergruppe nutzt Mendix Studio bzw. Mendix Studio Pro.
- Core Developer: Ist in der IT-Abteilung angesiedelt und hat mehrjährige Erfahrung im Bereich der Softwareentwicklung sowie der Integration von Anwendungen bzw. Daten in ein Anwendungsportfolio (Medix Evaluation Guide, 2019). Fungiert als Lead-Developer in einem ANDRITZ RAD-Team und arbeitet mit Mendix Studio Pro.
- Expert Developer: Ebenfalls in der IT-Abteilung beheimatet. Die Hauptaufgabe dieses Entwicklertyps ist es, die Mendix Plattformfunktionalität durch wiederverwendbare Komponenten zu erweitern. Mittels der Programmiersprache Java können Funktionalitäten in Microflows hinzugefügt werden, welche serverseitig ausgeführt werden und mittels der JavaScript Bibliothek React können Widgets erstellt werden, welche direkt am Client laufen. Die Arbeitsumgebung sind IDEs wie Eclipse oder IntelliJ.

Die gesamte Übersicht der Rollen und Verantwortlichkeiten, welche innerhalb des ANDRITZ RAD Services Verwendung finden, sind in Tabelle 7 ersichtlich. Abhängig vom Projekt, dessen Umfang und Komplexität ist es möglich, dass ein(e) MitarbeiterIn mehrere der beschriebenen Rollen

einnimmt bzw. das manche Rollen nicht besetzt werden. Spezielle Rollen, wie beispielsweise die des „Plattform Owners“, sind auf Projektebene nicht relevant.

Rolle	Verantwortlichkeiten
Product Owner (PO)	• Verantwortlich für den Projekterfolg (kann je nach Modell durch IT oder Fachbereich gestellt werden) • Koordination der Maßnahmen • Vorantreiben der Entwicklung • Abstimmung mit relevanten Stakeholdern
App Delivery Manager (ADM)	• Übersicht über alle in Entwicklung befindender Projekte • Laufende Statusreports an relevante Stakeholder • Gesamtübersicht Zeitplan / Budget
Enterprise Architekt (EA)	• Evaluation, ob Anforderung in das RAD-Portfolio passt • Erkennen möglicher Projektsynergien • Schnittstellenmanagement
Solution Architekt (SolA)	• Sicherstellen von Performance, Skalierbarkeit und Wartbarkeit auf Projektebene
Citizen Developer (CiDev)	• Erstellt selbstständig einfache Anwendungen • Mitentwicklung bei komplexen Anwendungen
Business Developer (BuDev)	• Datenkompetenz • Schnittstelle zwischen Fachbereich und IT
Core Developer (CoDev)	• Umsetzen von komplexen Features / Anwendungen • Lead Developer in einem RAD-Team
Expert Developer (ExDev)	• Erweitern der Mendix Plattformfunktionalität durch wiederverwendbare Komponenten (Java, React)
Operations (Ops)	• Monitoring des Anwendungsportfolios • Erstellung, Wartung, Integration der CI / CD Pipeline
User Experience Designer (UXD)	• Usability, angepasst auf verschiedenen Nutzergruppen je nach Projekt durch beispielsweise Personas, Customer Journeys, …
User Interface Designer (UID)	• Wartung, Aktualisierung und Weiterentwicklung des ANDRITZ Template • Projektspezifische Designanpassungen
Plattform Owner (PLO)	• Strategische Ausrichtung der Plattform (z.B. Re-Evaluation der Cloud Umgebung) • Verträge & Bedingungen mit Partnern • Aufbau und Weiterentwicklung des RAD-Teams • Koordination des Supports von externen Partnern • Definition von Standards und Best Practices zusammen mit Core Dev und Expet Dev

Tabelle 7: ANDRITZ RAD Rollen & Verantwortlichkeiten (eigene Darstellung)

4.2.8 **Aus- & Weiterbildung (QM, SO)**

Bevor beispielsweise ein Citizen Developer selbständig eine Anwendung entwickeln kann, ist es notwendig, ein grundlegendes Verständnis für die Entwicklungsumgebung sowie der darin bereitgestellten Funktionalitäten zu erwerben. Aus diesem Grund werden alle EntwicklerInnen, welche innerhalb des ANDRITZ RAD Services agieren, in einem Dev-Pool verwaltet. Voraussetzung, um in diesen Dev-Pool aufgenommen zu werden, und somit Zugang zu den benötigten ANDRITZ RAD-Werkzeugen (siehe Kapitel 4.2.2) zu erhalten, ist es, je nach angestrebter Rolle bzw. vorhandener Erfahrung, verschiedene Trainings zu absolvieren. Eine Auflistung der Klassifikation nach Erfahrung sowie dem daraus resultierenden Weiterentwicklungsangebot ist in Tabelle 8 ersichtlich.

Erfahrung: Nicht vorhanden

Kein Entwicklungswissen, nicht vertraut mit Konzepten und Techniken der Softwareentwicklung

Entwicklungsumgebung	• No-Code (Mendix Studio)
Weiterentwicklung	• Grundlegendes Training durch RAD Service, welches die Grundbegriffe, Konzepte und Techniken der SW-Entwicklung erläutert (~4h geblockt an einem Tag) • Weiterführendes Online-Training, um die erlernten Konzepte und Techniken in Beispielprojekten zu vertiefen (~ 12h bei freier Zeiteinteilung)

Basiswissen

Vertraut mit den grundlegenden Konzepten und Techniken der Softwareentwicklung, limitierte Praktische Erfahrung (erworben durch Trainings- oder Beispielprojekte)

Entwicklungsumgebung	• No-Code (Mendix Studio)
Weiterentwicklung	• Erweitertes Training durch RAD Service, welches die ANDRITZ RAD Best-Practices und Standards vermittelt (~4h geblockt an einem Tag) • Aufbauendes Online-Training, welches fortgeschrittenen Konzepte und Methoden erläutert (~12h bei freier Zeiteinteilung) • Sammeln von praktischer Erfahrung durch Mitarbeit in einfachen Projekten • Pair Programming

<table>
<tr><th colspan="2">Fortgeschritten</th></tr>
<tr><td colspan="2">Praktische Erfahrung beim Umsetzen von Softwareprojekten, benötigt noch Unterstützung beim Lösen komplexer Probleme

<table><tr><td>Entwicklungsumgebung</td><td>• Low Code (Mendix Studio Pro)</td></tr><tr><td>Weiterentwicklung</td><td>• Projektmitarbeit
• Mentoring durch ExpertInnen</td></tr></table></td></tr>
<tr><th colspan="2">Erfahren / Experte</th></tr>
<tr><td colspan="2">Setzt eigenständig Anforderungen um, benötigt keine Hilfestellung beim Lösen komplexer Probleme, Definition

<table><tr><td>Entwicklungsumgebung</td><td>• Low Code (Mendix Studio Pro)
• Eclipse, IntelliJ</td></tr><tr><td>Weiterentwicklung</td><td>• Externen Coachings bzw. Kurse, abgestimmt auf für die Person relevante Themenbereiche</td></tr></table></td></tr>
</table>

Tabelle 8: ANDRITZ RAD Aus- & Weiterbildung (eigene Darstellung)

4.2.9 Zusammenfassung ANDRITZ RAD CoE

Eine zusammenfassende Übersicht der in den letzten Kapiteln beschriebenen Funktionen des CoE ist in Abbildung 27 ersichtlich. Die einzelnen Umsetzungsteams bekommen durch die in gelb dargestellten Leistungen des CoE Unterstützung während der gesamten Projektlaufzeit. Die zur Verfügung stehenden Personen werden in einem Dev-Pool verwaltet, in welchem auch ein Aus- und Weiterbildungskonzept bereitgestellt wird.

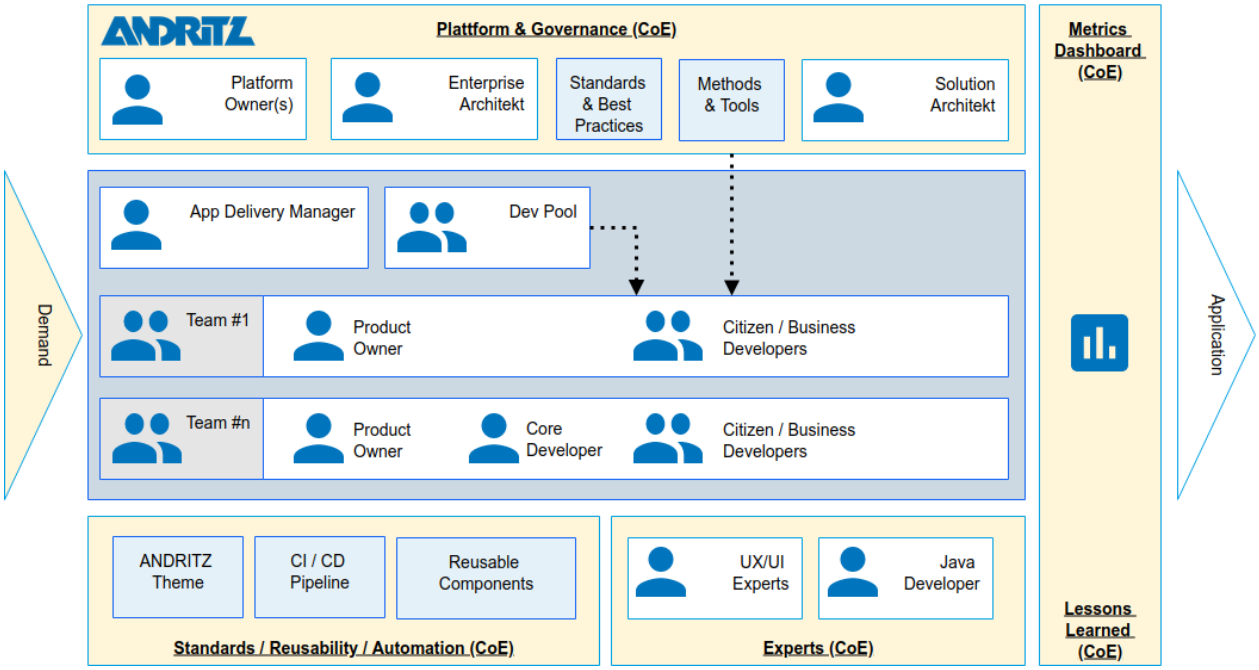

Abbildung 27: Andritz RAD CoE (eigene Darstellung angelehnt an Medix 2020)

Nachdem nun durch die einzelnen Bereiche des CoE die Grundlagen für die Abwicklung von Projekten im ANDRITZ RAD Service gelegt wurden, sollen diese im nächsten Schritt einem praktischen Test unterzogen werden. Der Ablauf der Fallstudie ist im folgenden Kapitel dokumentiert.

4.3 Ablauf der Fallstudie

Der Bedarf nach der in Kapitel 3.1.2 beschriebenen Anwendung, wurde durch den zuständigen Business Relationship Manager (BRM) an das RAD CoE herangetragen. Die Beschreibung der Anforderung bestand im Wesentlichen aus einer kurzen Beschreibung des Arbeitsablaufes eines Berechnungsingenieurs sowie der damals verwendeten, auf Excel basierenden, Lösung. Ziel war es, die Flut an Excel Dokumenten - es wurde ein Excel File je berechneter Maschine und je Version angelegt - durch eine zentral zugängliche Lösung zu ersetzen.

Um die Anforderungen besser zu verstehen und diese mittels dem in Kapitel 4.2.1.1 beschriebenen Template dokumentieren zu können, wurde im ersten Schritt ein Termin aufgesetzt, an dem zwei Key User aus der Fachabteilung sowie zwei KollegInnen des RAD CoE - ein Solution Architect sowie der Plattform Owner - beteiligt waren. In diesem Termin wurde der Arbeitsablauf, wie er zu diesem Zeitpunkt in der Fachabteilung stattgefunden hat, detailliert erläutert. Außerdem wurden seitens der Key User die gewünschten Funktionalitäten, welche die Anwendung bieten sollte, grob vorgestellt. Zu diesem Zeitpunkt stand bereits fest, dass es sich um eine webbasierte Anwendung handeln sollte. Aus diesem Grund sowie auf Basis der zu diesem Zeitpunkt bekannten Anforderungen wurde, zusammen mit dem Solution Architect, entschieden, diese Anwendung via Mendix zu realisieren und im ANDRITZ RAD Service umzusetzen.

Da seitens der Key User noch Unklarheiten bezüglich der genauen Workflows sowie der dafür benötigten Eingabemasken, Datenvalidierungen usw. bestand, wurde entschieden, diese Unschärfen durch einen Design Thinking Workshop auszumerzen. Dazu wurden im Vorfeld dieses Workshops, durchgeführt von einem externen Dienstleister, Interviews mit ausgewählten Key Usern abgehalten. Außerdem hatten die betreffenden Key User die Aufgabe, vorab definierte Arbeitsschritte durchzuführen, um die damalige Arbeitsweise aus unterschiedlichen Userperspektiven zu dokumentieren. Um solche Aufgaben in Zukunft selbstständig, innerhalb des RAD CoE, durchführen zu können, waren bei diesen Interviews sowie beim darauffolgenden Design Thinking Workshop mehrere RAD CoE KollegInnen als (teils) passive Beobachter beteiligt. Außerdem wurde eine Design Thinking Schulung für alle Mitglieder des RAD CoE organisiert und durchgeführt. Im eigentlichen Workshop selbst wurden die Erkenntnisse der Vorarbeiten aufbereitet und durchgesprochen. Um die für alle beteiligten Personen optimale Lösung bezüglich Workflows, Eingabemasken, Filter- /Suchkriterien usw. zu finden, wurden mittels Mendix durch einen RAD CoE Core Developer jeweils einfache Prototypen bzw. Klickdummies für bestimmte Funktionalitäten der Anwendung erstellt. Auf Basis der Rückmeldungen wurden diese Prototypen in mehreren Iterationen an die Bedürfnisse der Workshopteilnehmer angepasst, bis schließlich der gewünschte Arbeitsablauf gefunden und abschließend dokumentiert wurde.

Im Zuge dieses Workshops wurde der Bestandteil „User Design“ des in Kapitel 2.2.1 beschriebenen Vorgehensmodells „Rapid Application Development“ genutzt. Die Phase „Requirements Planning“ hatte ja bereits im Vorfeld stattgefunden.

Nachdem die Abläufe ab diesem Zeitpunkt für alle Beteiligten klar waren, wurde im nächsten Schritt die Art der Zusammenarbeit zwischen IT- und Fachabteilung definiert. Auf Basis der in Kapitel 4.2.1.1 vorgestellten Matrix wurde die Anwendung aufgrund ihrer Merkmale als „*High Complex / Low Exposure*“ eingestuft. Zwei der Key User aus der Fachabteilung, welche auch am Design Thinking Workshop teilgenommen hatten, konnten dafür gewonnen werden, sich aktiv an der Umsetzung der Anwendung zu beteiligen. Beide Kollegen konnten wenig, bis keine Erfahrung in Bezug auf Softwareentwicklung aufweisen, weshalb ein initiales Training durch das RAD CoE, wie in Kapitel 4.2.8 beschrieben, abgehalten wurde. In diesem Training wurde das nötige Basiswissen vermittelt, einerseits, um mit der grundlegenden Terminologie vertraut zu sein, andererseits, um die zu Grunde liegenden Konzepte der Entwicklung mit Mendix zu verstehen. Aufbauend auf dieses Training haben die beiden Kollegen selbstständig den empfohlenen weiterführenden online Kurs (Kapitel 4.2.8) absolviert.

Als Art der Zusammenarbeit wurde die Option „*Hybrides Projektmanagement*“ (Kapitel 4.2.1.1) gewählt, mit jeweils einem Product Owner auf IT- und Fachbereichsseite. Dies entspricht dem in Kapitel 2.5.2 beschriebenen Faktor „geteiltes Führen“. Es waren auf Fachbereichsseite sowie durch Kollegen des RAD CoE somit folgende Rollen im Projektteam besetzt, welche in Tabelle 9 ersichtlich sind:

Rolle Fachbereich (Ort)	Verantwortlichkeit	Zuteilung[15]
Business Developer (CHE)	PO auf Fachbereichsseite, Testen	50%
Citizen Developer (DEU)	Entwicklung, Testen	50%
Rolle RAD CoE		
Product Owner (AUT)	PO auf IT-Seite, Testen	100%
Core Developer (AUT)	Lead Developer	100%
Solution Architect (FIN)	Performance, Erweiterbar- und Wartbarkeit der Anwendung	Bei Bedarf (Code Review Meetings)
Expert Developer (AT)	Entwicklung eines wiederverwendbaren Widgets zur BOM Darstellung	Für die Dauer der Widget Entwicklung

Tabelle 9: Rollen der Projektmitglieder während der Fallstudie (eigene Darstellung)

Da innerhalb der ANDRITZ AG BOM Strukturen häufig zum Einsatz kommen und dafür noch kein Widget verfügbar war, welches in Mendix verwendet werden konnte, wurde entschieden, ein wiederverwendbares Widget zu entwickeln, welches die Anforderungen (beispielsweise Drag & Drop von Elementen über die gesamte BOM hinweg, das Klonen, Löschen und Editieren von Elementen sowie die Versionierung von Kompletten BOM Strukturen) dieser und zukünftigen

[15] Zuteilung zum Projekt in % der verfügbaren Arbeitszeit

Anwendungen, welche auf BOM Strukturen basieren, erfüllt. Die Entwicklung dieses Widgets wurde durch einen RAD CoE Expert Developer realisiert.

Auf den Einsatz eines / einer UI ExpertIn wurde aufgrund der lediglich ANDRITZ internen Nutzung der Anwendung verzichtet – es wurde das vom RAD CoE zur Verfügung gestellte und als Startpunkt für Mendix Applikationen verfügbare, ANDRITZ Design Template verwendet. Außerdem wurde auf die Rolle eines Scrum Masters verzichtet, diese Aufgaben hat der Product Owner auf IT-Seite bei Bedarf übernommen.

Wie bereits erwähnt, wurde während des Design Thinking Workshops, durch das Erstellen von Prototypen und deren ständiger Verbesserung aufgrund der Rückmeldung der Key User, bereits mit Teilen von Rapid Application Development als Umsetzungsmethode gearbeitet. Die noch umzusetzenden Phasen laut Rapid Application Development wären „Construction“ und „Cutover“ (siehe Kapitel 2.2.1) gewesen. In Anbetracht der Tatsache, dass die Anforderungen während des Design Thinking Workshops jedoch sehr klar und detailliert erfasst wurden und der Kunde direkt bei der Umsetzung (Implementierung) des Projektes beteiligt war - aufgrund dessen keine weiteren Iterationen bzw. Prototypen durch Änderungswünsche mehr zu erwarten waren (die Unsicherheit der Anforderungen wurde eliminiert) -, wurde entschieden, die verbleibenden o.a. Phasen mit einer anderen Umsetzungsmethode abzuschließen. Auf Basis der in Kapitel 4.2.1.3 beschriebenen Einflussgrößen wurde beschlossen, eine agil-hybride Umsetzungsmethode zu wählen.

Angewandt wurde ein Mix aus Kanban und Scrum - diese Kombination ist als Scrumban bekannt. Dabei sind beispielsweise keine fixen Sprintzyklen, in denen vorab definierte Stories umgesetzt werden, enthalten. Stattdessen werden so genannte Work In Progress (WIP) Limits verwendet und auf einem Kanban Board abgebildet. Diese Limits beschränken die Anzahl der in Umsetzung befindlichen Aufgaben je Arbeitsschritt. Das bietet den Vorteil, das ToDo`s jederzeit Situationsabhängig angepasst werden können. Außerdem wurde auf das in Scrum gängige Schätzen von Stories verzichtet, da die wenig erfahrenen EntwicklerInnen der Fachabteilung in ihrem ersten Projekt ohne zu viel Druck arbeiten sollten.

So wurde wie bereits erwähnt, ein Backlog bestehend aus User Stories erstellt, welcher priorisiert am Kanban Board in der To-Do-Spalte angezeigt und abgearbeitet wurde. Dabei wurde darauf geachtet, dass die weniger Erfahrenen Citizen / Business Developer nur Tasks zur Umsetzung erhalten haben, welche aufgrund ihrer begrenzten Verfügbarkeit von rund 50% (gerechnet auf die reguläre Arbeitszeit) keine kritischen Abhängigkeiten enthalten und somit die Entwicklung nicht unnötig verzögern haben. Von den Kollegen aus der Fachabteilung wurden, verglichen mit dem RAD Core Developer, der Hauptteil der notwendigen Tests abgedeckt. In vielen Fällen konnten die EntwicklerInnen der Fachabteilung dabei gefundene Fehlfunktionen (Bugs) selbstständig, durch Hilfe der Dokumentation, ausbessern und den Task somit abschließen.

Aufgrund der moderaten Projektgröße sowie der „kurzen“ Projektlaufzeit von rund 8 Wochen wurde darauf verzichtet, eine CI / CD Pipeline zu erstellen und in das Projekt zu integrieren. Lediglich die Backups der Datenbank wurden mittels Jenkins automatisiert. Neue Features bzw. Anpassungen oder Bugfixes wurden, nachdem diese lokal getestet wurden, auf den Teamserver committed. Für jedes zu entwickelnde Feature wurde im Mendix integrierten

Versionierungssystem ein eigener Branch erstellt, welcher nach einem Review durch den RAD Core Developer entweder mit dem Main Branch zusammengeführt (gemerged) oder mit Änderungsanweisungen zurückgewiesen wurde.

Die Qualität der Anwendung wurde darüber hinaus in unregelmäßigen Abständen durch den Solution Architect zusammen mit dem Lead Developer bewertet, indem Microflows sowie das Datenmodell einem Review unterzogen wurden. Spezielles Augenmerk wurde dabei auf die Themen Performance sowie Erweiterbarkeit gelegt. Im Zuge solcher Reviews wurden, falls verbesserungswürdige Bestandteile innerhalb der Anwendung entdeckt wurden, dementsprechende JIRA Tickets erstellt, welche mit dem Label „Best Practice“ oder „Standard“ versehen wurden. Innerhalb des RAD CoE wurden diese Tickets anschließend diskutiert und in weiterer Folge, entweder als Best Practice oder Standard, in die verfügbare Dokumentation mit aufgenommen oder, falls zu spezifisch oder irrelevant, verworfen.

Mehrmals täglich - die nötige Abstimmung erfolgte im dafür vorgesehenen MS-Teams „Team“ -, wurde von einem Core Developer auf die Testumgebung deployed. Nach erfolgreichen Tests auf dieser Umgebung wurden die Features schließlich in die Produktionsumgebung gehoben. Auf eine Qualitätsumgebung, auf der normalerweise der Kunde testet, wurde verzichtet, da durch das Mitwirken der Fachabteilung am Entwicklungsprozess die nötigen Tests seitens des Auftraggebers bereits auf der Testumgebung durchgeführt wurden.

Aus Scrum wurde das Daily Standup übernommen, in dem täglich ein kurzes Statusupdate durchgeführt wurde, an dem sämtliche Projektmitglieder teilgenommen haben. Darin wurden die drei Punkte „Was habe ich seit dem letzten Standup gemacht?“, „Was plane ich bis zum nächsten Standup zu machen?“ sowie „Was behindert mich bei der Umsetzung?“ besprochen. Aufgezeigte Hindernisse, auch Impedimentes genannt, wurden vom Product Owner notiert und so gut als möglich beseitigt. Neben den Daily Standups, wurde jede zweite Woche eine Retrospektive durchgeführt, in der die generelle Zusammenarbeit, die aufgetretenen Hindernisse sowie der Projektfortschritt besprochen und beurteilt wurden.

Wie bereits in Kapitel 4.2.2.1 beschrieben, wurde die gesamte Kommunikation in einem eigens für das Projekt erstellten MS-Teams „Team“ abgehalten. Ebenso wurde „Quizzbreaker“ als virtuelle Team Software genutzt.

Das Projekt konnte nach rund 8 Wochen in den Produktionsbetrieb übergehen. Seitdem werden, seitens der beiden Kollegen der Fachabteilung nötige Weiterentwicklungen, selbständig durchgeführt und getestet. Bei Bedarf werden Kollegen des RAD CoE wie in Kapitel 4.2 beschrieben, hinzugezogen. Das deployen von neuen Versionen der Anwendung findet weiterhin zentral durch das RAD CoE statt.

4.4 Ergebnisse der ExpertInneninterviews

Dass es sich bei den vier interviewten Personen ausschließlich um männliche Kollegen handelt, wird im folgenden Kapitel entsprechend auf ein Gendern der Begriffe ExpertIn, KollegIn usw. verzichtet.

Auf Basis der in Kapitel 3.2.4 beschriebenen Methode der strukturierten Inhaltsanalyse, wurden die transkribierten Interviews anhand von vordefinierten Kategorien analysiert. Diese Kategorien orientieren sich, wie auch der Interviewleitfaden, grob am Software-Lebenszyklus. Es wurden dabei jedoch einzelne Schritte des SDLC in einer Kategorie zusammengefasst (beispielsweise sind die SDLC Kategorien „Identification“, „Inception“ und „Initiation“ in der Kategorie Anforderungsmanagement zusammengefasst). Ziel war es herauszufinden, in welchen Phasen Probleme bzw. Hindernisse aufgetreten sind, um entsprechende Anpassungen vornehmen zu können, um die aufgetretenen Hürden bei den folgenden Projekten zu vermeiden. Es wurden insgesamt 4 Hauptkategorien gebildet, die Einträge unter diesen Kategorien spiegeln den Inhalt nach Themenblöcken wider. Tabelle 10 zeigt eine Übersicht der Kategorien und der darin enthaltenen Themenblöcke.

Anforderungsmanagement / Projekt Vorphase
- Interviews / Beobachtung
- Design Thinking Workshop
- Aus - und Weiterbildung (Antworten nur von Kollegen der Fachabteilung)
Projekt Umsetzung
- Hilfsmittel / Werkzeuge / Einflussfaktoren
- Hindernisse / Einflüsse von außen
- Kommunikation: Kanäle und Regeln
Betrieb (Antworten nur von Kollegen der Fachabteilung)
- Zusammenarbeit RAD CoE – Fachbereich nach Abschluss des laufenden Projektes
Sonstiges / Vergangene Erfahrungen
- Nicht bedachte Aspekte / Probleme / Sichtweisen aus anderen Projekten / Projekterfahrungen einfließen zu lassen - Sonstige Anmerkungen zum Projektverlauf, welche nicht in eine der o.a. Kategorien fallen

Tabelle 10: Kategorien der strukturierten Inhaltsanalyse (eigene Darstellung)

Um die Haupt- bzw. Unterkategorien mit Aussagen aus den Experteninterviews in Verbindung zu bringen, werden Ankerbeispiele in kursiver Schrift mit Informationen zum betreffenden Interview dargestellt. Anmerkungen in den Zitaten, beispielweise wenn Namen anonymisiert wurden, sind in eckigen Klammern [] enthalten. Ausgelassene Teile des Interviews, welche keinen Mehrwert im angefügten Ankerbeispiel bringen, sind mit (…) gekennzeichnet.

4.4.1 **Anforderungsmanagement / Projekt Vorphase**

Die befragten Experten sind sich einig, dass die in Kapitel 4.1.1 beschriebene Phase der „Anforderungsanalyse“ starken Einfluss auf den Projekterfolg hatte. Speziell der Design Thinking Workshop sowie die vorab durchgeführten Interviews und Klicktests mit ausgewählten Key Usern des Fachbereiches, haben nach Meinung der Experten die Basis für eine erfolgreiche Projektrealisation gelegt.

> *Die Interviews und Klicktests, die vor dem Workshop stattgefunden hatten waren meiner Meinung nach die Basis für den erfolgreichen Workshoptag (…) Der Workshop selbst hat dem ganzen dann noch ein Level an Detail hinzugefügt (…) Der Zeitaufwand war überschaubar, die Ergebnisse dafür ausgezeichnet, meiner Meinung nach (…) die Anforderungen müssen klar kommuniziert und dokumentiert werden, sonst zieht sich die Umsetzungsphase unnötig in die Länge. (Experte 1)*

Obwohl Experte 3 den Workshop auch positiv in Bezug auf die genaue Spezifikation der Anforderungen sieht, wird die Verhältnismäßigkeit in Frage gestellt:

> *Es ist natürlich wichtig und richtig, dass in der ersten Phase des Projektes alles Nötige unternommen wird, um die Anforderungen so genau als nur möglich zu dokumentieren (…) macht so ein Workshop natürlich Sinn (…) Meiner Erfahrung nach, erspart man sich viel Zeit und Ärger während der Projektumsetzung, wenn die Anforderungen zu Beginn genau spezifiziert und dokumentiert werden (…) in unserem Fall (…), weil die Kollegen aus Deutschland [Fachabteilung] auch direkt während der Umsetzung des Projektes beteiligt waren, hätte man sich das aber auch sparen können. Offene Fragen und Unklarheiten können in so einem Fall, während der Umsetzungsphase, auch gut im Standup geklärt werden. (Experte 3)*

Experte 2 bewertet zusätzlich die während des Design Thinking Workshops entwickelten Prototypen, welche beispielsweise die Eingabemasken und Seitenabläufe auf einem ANDRITZ Bildschirm (Standard Größe von 24 Zoll) dargestellt haben, als hilfreich:

> *(…) gut gefunden, dass [RAD] immer wieder Prototypen für kleine Teile der Anwendung erstellt hat (…) Es macht einen Unterschied, ob man etwas auf Papier entwickelt oder etwas vor sich hat, das man direkt bedienen kann. Gerade die Pageflows, das Design der Eingabemasken auf einem Bildschirm mit der Größe meines Arbeitsbildschirms usw. bringt schon Vorteile. Man kann sich so einfach besser vorstellen, wie es ist mit der fertigen Anwendung zu arbeiten. (Experte 2)*

Der Einflussfaktor „Team“ (Kapitel 4.1.1) beinhaltet wie beschrieben auch den Punkt Aus- und Weiterbildung. Im Zuge des Projektes wurden vor dem eigentlichen Projektstart zwei Schulungen für die Kollegen der Fachabteilung durchgeführt – eine ANDRITZ interne Schulung, welche die Grundlagen der modellgetriebenen Softwareentwicklung mittels Mendix vermittelt sowie ein aufbauendes Selbststudium, welches die Kollegen des Fachbereichs eigenständig durchgeführt haben. Die beiden befragten Experten 1 und 2 erachten die Schulungsmaßnahmen als notwendig und sinnvoll, jedoch wurde der Aufbau bzw. die Abfolge der Maßnahmen in Frage gestellt:

> *Die Inhalte werden logisch aufbauend vermittelt, dass man sich auch ohne Vorkenntnisse die Grundlagen schnell aneignen kann (…) es wäre sinnvoller (…) zuerst das Selbststudium,*

> *danach die Schulung von euch [RAD]. Dabei können dann offene Fragen, die während des Selbststudiums aufgetreten sind, geklärt werden. (Experte 1)*
>
> *Der Crashkurs durch [RAD] war kurz und knackig, länger hätte er für mich aber auf keinen Fall dauern dürfen. Es ist schon sehr viel Information, die man da in 4 Stunden aufnehmen muss, vor allem wenn man vorher wenig oder gar nichts mit Objektorientierung usw. zu tun hatte. Man kann sich das Wissen aber auch komplett selbstständig via Online Training, direkt bei Mendix holen (…) Deswegen denke ich, die Reihenfolge wäre umgekehrt sinnvoller. Zuerst das Selbststudium im eigenen Tempo, danach ein aufbauender Kurs mit euch [RAD] (…) ohne ein Training, in welcher Art auch immer, würde die Zusammenarbeit in der Form, wie wir sie hatten, nicht funktionieren. (Experte 2)*

Zusammengefasst erachten die befragten Experten ein vorab durchgeführtes online Selbststudium, welches durch ein vom RAD Service abgehaltenes Aufbautraining abgerundet wird, für sinnvoller. Im Aufbautraining können ggf. offene Fragen, die während des Selbststudiums aufgetreten sind, beantwortet werden. Außerdem besteht die Möglichkeit, fortgeschrittene Konzepte, welche evtl. für das folgende Projekt benötigt werden, zu behandeln. Die behandelten Inhalte der beiden Schulungen waren, laut Aussagen der Experten, für die folgende Phase der Projektumsetzung ausreichend:

> *Man kennt die Grundlagen, spricht eine Sprache und kann dadurch bei Problemen auch effektiver kommunizieren. (Experte 1)*
>
> *Für die Aufgaben, die ich im Projekt zu machen hatte, waren die Themen der Kurse im Großen und Ganzen ausreichend. (Experte 2)*

Ein mögliches Problem sehen die beiden befragten Experten hinsichtlich der parallel zum Tagesgeschäft laufenden Schulungsmaßnahmen. Wird das Training zu oft bzw. zu lange durch „Störungen“ unterbrochen, besteht die Gefahr, dass wesentliche Inhalte unzureichend verinnerlicht werden. Vorgeschlagen wurde von Experte 1 die beiden Trainings geblockt an 2 Tagen in einem Besprechungsraum durchzuführen:

> *Man muss sich ein Zeitfenster einplanen, das exklusiv für das Training vorgesehen ist. Der Onlinekurs ist, wenn ich mich richtig erinnere, mit 12h angeschrieben. Das könnte man, wenn man 2 Tage blockt und das RAD Training direkt anschließt, kompakt und effektiv durchführen, vielleicht sogar mit allen Teilnehmern in einem Besprechungsraum, damit man sich dann gegenseitig helfen kann. (Experte 1)*

Aufgrund der fehlenden Erfahrung der beiden Experten der Fachabteilung mit Vorgehensmodellen, welche im Bereich der Softwareentwicklung zum Einsatz kommen, wurde aufgrund der Aussagen in den Experteninterviews entschieden, außerdem ein agiles Grundlagentraining in den in Kapitel 4.2.8 beschrieben Aus- und Weiterbildungskatalog aufzunehmen.

4.4.2 **Projekt Umsetzung**

Der Kommunikationsfaktor „geteiltes Führen**“**, welcher in Kapitel 2.5.2 beschrieben wurde, wurde laut Aussagen der betroffenen Experten, gerade in Bezug auf das zur Verfügung stehende Budget sowie auf die Verteilung der Aufgaben bezogen, als hilfreich eingeschätzt. Experte 3 hat dazu folgendes Statement abgegeben:

> *Als PO muss ich ja auch darauf schauen, dass das verfügbare Budget nicht überschritten wird. Wenn du da jemanden von (…) Kundenseite hast, der weiß, dass zusätzliche Features Geld kosten und das auch abschätzen kann, erleichtert das die Zusammenarbeit schon um einiges (…) es entstehen kurze Wege und dadurch schnelle Antwortzeiten bei Fragen oder Unklarheiten – das war durch die enge Kooperation ein Vorteil. (Experte 3)*

Experte 1 bewertet den Faktor „geteiltes Führen“ ebenfalls positiv. Speziell die dadurch in der Fachabteilung entstandene Möglichkeit, eigenständig Entscheidungen treffen zu können, wird als Vorteil betrachtet. Das folgende Statement von Experte 1 kann sowohl mit dem Faktor „geteiltes Führen“ als auch mit dem Faktor „Feedback“ in Verbindung gebracht werden:

> *Sehr positiv war die Zusammenarbeit mit [Experte 3]. Wir haben sozusagen als doppelspitze agiert (…) wir haben uns teilweise sehr gut ergänzt, zum Beispiel beim Schreiben der User Stories. Meine Funktion im Projektteam als „Business Developer“ war ja eine Art Brückenbauer zwischen IT und Fachbereich. Ich denke, das hat gut geklappt, [Experte 2] konnte sich dadurch voll und ganz auf die Aufgabe der Entwicklung und des Testens konzentrieren (…) Die meisten Fragen (…) konnten Großteils von mir direkt beantwortet werden, ohne das [Experte 3] aktiv werden musste. (Experte 1)*

Der bereits erwähnte Faktor „Feedback“ (Kapitel 4.1.1) hat, laut Aussagen der Experten, durch die direkte Zusammenarbeit der Bereiche IT und Fachabteilung, positiv verkürzenden Einfluss auf die Projektlaufzeit. Durch die direkte Beteiligung des Fachbereichs an der Projektumsetzung konnte die Auslieferung des Produktes, laut Aussage der Experten 3 sowie 1, deutlich beschleunig werden, da die ansonsten nötigen Feedbackschleifen verkürzt werden konnten:

> *Wir haben uns Zeit erspart, weil der Kunde direkt an der Umsetzung beteiligt gewesen ist. Normalerweise lieferst du ja nach einer Iteration eine neue Version der Anwendung an den Kunden. Der testet dann und gibt Feedback, meldet Bugs, kritisiert den Workflow usw. Das war diesmal nicht der Fall, da der Kunde durch den ständigen und engen Kontakt und dem daraus folgenden kontinuierlichen Feedback durchgehend Einfluss auf die Anwendung hatte. Die Kollegen haben auch relativ viel getestet und konnten den Großteil der gefundenen Defects selbständig beseitigen. (Experte 3)*

Experte 2 sieht in Bezug auf die verkürzten Feedbackschleifen hauptsächlich das vom Fachbereich durchgeführte Testen und das darauf aufbauende selbstständige Beheben von Fehlern als entscheidenden Faktor:

> *Wir [Experte 1 und Experte 2] haben ja viel getestet von dem was ihr [RAD CoE] entwickelt habt. Die Fehler, die wir dabei entdeckt haben, konnten wir zum Teil selbst beheben und haben uns dadurch viel Zeit erspart. (Experte 2)*

Während der Umsetzungsphase sind auch verschiedene technische Hindernisse aufgetreten, welche von den Experten währen der Interviews als hinderlich eingestuft wurden. Kleinigkeiten, wie beispielsweise Merge-Konflikte, konnten anhand der verfügbaren Dokumentation von den betroffenen Entwicklern selbstständig gelöst werden.

> *Da gibt es einen guten Eintrag im FAQ Bereich, der sich mit Merge Problemen beschäftigt. Mendix macht es einen dabei aber auch relativ einfach. Das konnte ich also selbst lösen. (Experte 2)*

Die restlichen, größere technische Hindernisse, welche das gesamte Projekt betroffen haben, wurden von den Kollegen des RAD CoE bearbeitet und sind unabhängig vom in dieser Arbeit entwickelten Vorgehensmodell. Der Vollständigkeit wegen werden sie trotzdem kurz beschrieben. Während der Umsetzungsphase konnten zwei solcher technischen Hindernisse beseitigt werden. Da die Kollegen des RAD CoE diese Probleme jedoch nicht selbstständig lösen konnten, wurde der Mendix Support, wie in Kapitel 4.2.2.1 beschrieben, durch den Plattform Owner kontaktiert. Zum einen war es durch ein aus dem Mendix App Store bezogenes, von Mendix entwickeltes, Widget nicht möglich, eine funktionierende Schnittstelle zwischen Mendix Applikation und ANDRITZ SAP herzustellen. Dieses Problem wurde, koordiniert vom Plattform Owner des RAD-Services sowie der Hilfe vom Mendix- und SAP-Support, gelöst. Das Problem konnte auf ein fehlerhaftes Verhalten des Widgets zurückgeführt werden und wurde von Mendix durch das zur Verfügung stellen einer neuen Version des Widgets behoben.

Das zweite Problem wurde durch einen Versionskonflikt zwischen dem verwendeten Mendix Studio Pro und dem native Builder, welcher für das Erstellen der mobilen Pakete (. ipa und .apk) für Android und iOS zuständig ist, verursacht. Dieses Problem konnte ebenso durch die Hilfe des Mendix Support gelöst werden. Die gesamte Koordination und Abwicklung der Probleme wurde, wie bereits erwähnt, zentral vom RAD Service aus gesteuert, was laut Aussagen der Experten des Fachbereichs, positiv bewertet wurde.

Ein weiteres technisches Hindernis lässt sich auf das erstellte Vorgehensmodell und den darin definierten Werkzeugen zurückführen und wurde von Experten 2 aufgrund der limitierten Funktionalität von Mendix Studio erwähnt:

> *(…) zu Beginn war ja vereinbart, dass ich als Citizen Developer mit Mendix Studio arbeite. Das hat nicht gut funktioniert, da ein Großteil der nützlichen Funktionen nicht enthalten sind (…) Problem war der fehlende Debugger, ich konnte somit nicht testen. Das hat die online Version [Mendix Studio] für mein Aufgabengebiet ungeeignet gemacht. (Experte 2)*

Die in Kapitel 4.2.2 beschriebenen und im RAD Service zur Verfügung stehenden Werkzeuge, werden nach Rückmeldung der Experten des Fachbereichs um das Tool pgAdmin (PostreSQL Datenbank) erweitert. Damit soll es ermöglicht werden, für die lokale Entwicklung dieselbe Datenbank zu verwenden, welche auch in der Cloud Umgebung zur Verfügung steht. Dadurch kann der Datenstand, welcher am Test- oder Produktivsystem verfügbar ist, durch einen Datenbank Dump auch lokal, beispielsweise für Testzwecke, verwendet werden.

> *Es macht für das Testen aber schon Sinn, wenn man einen Datenstand hat, der auf dem Test- oder Produktivsystem zumindest ähnlich ist. Daten ständig manuell zu befüllen ist ein unnötiger Zeitfresser. (Experte 2)*

Abgesehen von den erwähnten Problem mit Mendix Studio und der fehlenden Verfügbarkeit einer PostgreSQL Datenbank für den Citizen Developer, wurden die verfügbaren Werkzeuge jedoch als ausreichend für die Projektabwicklung angesehen.

Übereinstimmend wurde die bereits erwähnte Dokumentation in Confluence, bestehend aus FAQ, Best-Practices und Standards als nützlich, wenn auch im derzeit vorhandenen Ausmaß als zu wenig umfangreich bewertet. Experte 3 regt an, den Dokumentationsbereich, um eine Art RAD Leistungskatalog zu erweitern, in der die umgesetzten Anwendungen von der Anforderung bis zum fertigen Produkt dokumentiert werden. Außerdem sieht er in der Dokumentation die Voraussetzung, für ein selbstständiges Arbeiten mit RAD Werkzeugen, im Fachbereich.

> *Der Doku Bereich ist ja noch im Aufbau, da fehlen noch Themenbereiche (…) Derzeit ist der ganze Bereich eher auf technische Hilfe ausgelegt (…) ein wichtiger Teil der fehlt, ist das ganze Thema UI / UX (…) auch die Ergebnisse der Design Thinking Workshops sollten wir aufbereiten und zugänglich machen (…) als RAD Leistungskatalog zum Beispiel (…) je mehr Material zur Verfügung steht, desto einfacher kann der Fachbereich selbständig agieren (…) schön strukturiert aufgebaut (…) wie eine Wissensdatenbank, in der ich die Themen auch finde nach denen ich suche (…) Ohne umfangreiche Dokumentation wird eine selbstständige Umsetzung im Fachbereich, denke ich, sehr schwierig. (Experte 3)*

Für ein eigenständiges Umsetzen von Projekten innerhalb der Fachabteilung, sieht Experte 2 die derzeit verfügbare Dokumentation, als zu gering an.

> *Die Artikel, die jetzt schon verfügbar sind, sind sehr hilfreich (…) Wenn in Zukunft aber Projekte direkt in der Fachabteilung umgesetzt werden sollen, müssen noch deutlich mehr Themenbereiche abgedeckt werden (…) Ich habe ja dann keine Möglichkeit mehr beim Daily Standup diese Probleme mit einem erfahrenen Entwickler zu besprechen und jedes Mal den RAD Service zu kontaktieren, finde ich eher mühsam. (Experte 2)*

Experte 1 sieht als positiven Punkt jedoch auch die schnelle Erweiterung der verfügbaren Dokumentation, welche nach einem Review des Themas innerhalb des RAD Services stattfindet:

> *Der Prozess, dass die Dokumentation durch Review Ergebnisse oder durch anraten von Projektmitgliedern ständig erweitert wird, ist gut und es passiert meist relativ schnell während der Projektphase, wenn die Themen aktuell sind. (Experte 1)*

Der Großteil des Testens wurde von den Kollegen des Fachbereichs erledigt, was laut Aussage von Experte 2, einerseits zur Weiterentwicklung eines Citizen Developers selbst beitragen kann, andererseits sicherstellt, dass die gewünschte Anwendung auch den Bedürfnissen der Auftraggeber entspricht:

> *(…) wird die Lösung ja für uns und mit uns entwickelt, wir müssen dann am Ende des Tages damit arbeiten. Ich denke deswegen ist man beim Testen vielleicht auch etwas genauer und akribischer (…) Ich denke auch, dass (…) eine gute Möglichkeit ist, das erworbene Wissen zu vertiefen (…) Ich habe dadurch viel gelernt, das Hilft mir jetzt beim Weiterentwickeln der Anwendung. (Experte 2)*

Experte 1 sieht den Vorteil ebenfalls darin, dass der Fachbereich die Anwendung sehr detailliert kennenlernt und das Wissen nicht nur auf die Funktionen beschränkt bleibt, welche vom Fachbereich selbst implementiert werden.

> *Das war sinnvoll, wir haben dadurch die Anwendung in einem Detailgrad kennengelernt, der nicht möglich gewesen wäre, wenn wir nicht so viel getestet hätten. Das Hilft uns bei der Weiterentwicklung, die wir jetzt ja in der Abteilung selbst machen (…) wir haben ja das größte Interesse daran, dass die Anwendung so funktioniert, wie wir uns das vorgestellt haben. (Experte 1)*

Ebenfalls positiv bewertet wurde die Tatsache, dass die Kollegen der Fachabteilung aufgrund ihrer geringeren Verfügbarkeit nur Arbeitspakete umzusetzen hatten, von denen keine weiteren Arbeitspakete abhängig waren. Dies entspricht dem in Kapitel 2.5.2 beschriebenen Faktor „Koordination“. Zum einen konnte so vermieden werden, das Projekt unnötig zu verzögern, zum andern fühlten sich die Entwickler der Fachabteilung dadurch in ihrem ersten IT-Projekt nicht unnötig unter Druck gesetzt.

> *(…) haben wir vereinbart, dass nur Aufgaben von uns übernommen werden, die unabhängig von andern umgesetzt werden können und von denen auch keine anderen Aufgaben abhängen. Wir hatten somit wenig Druck, weil wir nicht das Gefühl hatten, dass jemand auf unser Arbeitspaket wartet beispielsweise. Das fand ich gut in Anbetracht meiner geringen Verfügbarkeit und geringen Entwicklungswissen. (Experte 1)*

Das Projekt wurde mit Scrumban, einer hybriden Projektmanagement Methode, welche Einflüsse aus Scrum und Kanban enthält, durchgeführt. Auf eine generelle Bewertung dieser speziellen Umsetzungsmethode wurde verzichtet, da die Wahl der Methode laut Kapitel 4.2.1.3 sich von Projekt zu Projekt unterscheiden kann. Die Möglichkeit, situationsabhängig die passende Umsetzungsmethode zu wählen, wird von Experte 3 grundsätzlich positiv betrachtet. Allerdings ist dieser auch der Meinung, dass sich die eingesetzten Methoden in Zukunft nicht sonderlich voneinander unterscheiden werden:

> *In der Theorie klingt das gut, in der Praxis werden wir aber, denke ich, in den allermeisten Fällen entweder Scrum oder Kanban einsetzen – oder eben einen Mix aus diesen beiden. Ich kann mir nicht vorstellen, dass wir mit Mendix Projekte in einer Größenordnung und Komplexität umsetzen, die den Einsatz eines anderen Vorgehensmodells nötig machen. (Experte 3)*

Da die Experten 1 und 2 der Fachabteilung, wie bereits in Kapitel 4.4.1 erwähnt, keine Erfahrung mit anderen Umsetzungsmethoden aufweisen können, wurde auf eine Bewertung dieser ihrerseits verzichtet.

Als für das Projekt hinderliche Einflüsse von außen, wurden von den Experten 1 und 2 des Fachbereichs, die prozentuale Aufteilung von jeweils 50% auf Projektarbeit und Tagesgeschäft genannt. Experte 3 erwähnt darüber hinaus noch die mögliche Projektverzögerung, die sich aufgrund der technischen Hindernisse und der damit verbundenen Abhängigkeit zum Mendix Support ergeben hat.

> *Dadurch entsteht natürlich eine Abhängigkeit zum Support von Mendix, der für die Lösung der Probleme zuständig ist. Wenn die Lösung länger gedauert hätte, wäre ein Projektverzug auch nicht unrealistisch gewesen. Sowas kann natürlich immer passieren, es ist aber ärgerlich, wenn es Fehler in vorgefertigten Komponenten gibt, für die du selber nichts kannst und die dann den Zeitplan gefährden. (Experte 3)*

Die im Projekt verfügbaren Kommunikationskanäle, bestehend aus MS-Teams für die projektinterne Kommunikation, Yammer für die Kommunikation innerhalb des gesamten RAD Services, über Projektgrenzen hinweg, sowie dem Tool Quizzbreaker zur Stärkung des Teamgefühls, wurden von den befragten Experten als unterschiedlich sinnvoll bewertet.

MS-Teams als Kommunikationskanal wurde dabei durchwegs positiv beurteilt. Dabei ist anzumerken, dass die Kollegen des Fachbereichs die MS-Teams Lizenz aufgrund ihrer Projektbeteiligung erhalten haben, da der Rollout des Tools in Österreich und Deutschland, außerhalb der IT, aufgrund einer Intervention des Betriebsrates, gestoppt wurde. Für die Kollegen des Fachbereiches hat es im Vorfeld des Projektes ein spezielles „MS-Teams Onboarding“ gegeben, welches von den Kollegen des Enterprise-Content-Management Teams (ECM) abgehalten wurde.

> *Es ist aber, meiner Meinung, nach sehr komfortabel, die gesamte projektbezogene Kommunikation und alle notwendigen Informationen von einem zentralen Ort aus verfügbar zu haben (…) Ich öffne Teams und kann im Projekt Channel gleich zu den User Stories in Confluence springen, habe die Chat Historie dokumentiert, alle benötigten Personen in einem Raum. (Experte 1)*

> *Es macht das Handling, speziell wenn man mehrere Projekte parallel bearbeitet, um einiges einfacher und übersichtlicher (…) Das [MS-Teams] ist auch Voraussetzung für die Zusammenarbeit, mit Outlook und Skype kannst du nicht so gezielt innerhalb eines Projektes kommunizieren. (Experte 3)*

Die Idee, Yammer als soziales Firmennetzwerk zu verwenden und damit alle im RAD Umfeld beteiligten Personen zu vernetzen, damit den Wissensaustausch zu fördern, wird zwar als sinnvoll eingestuft, tatsächlich verwendet wurde das Tool von den Kollegen der Fachabteilung während dem Projekt, jedoch nur in einem geringen Ausmaß. Als Grund nannten die befragten Experten, das derzeit zu wenige Themen verfügbar sind und dementsprechend auch wenig Interaktion zwischen den Personen im RAD Umfeld auf der Plattform stattfindet.

> *Derzeit ist darin ja auch noch nicht viel los. Ich denke, das ist auch ein Grund, warum das für mich eher uninteressant war (…) Wenn das Ganze etwas an Fahrt aufnimmt, kann ich mir aber schon vorstellen, dass dort interessante Themen aufkommen können und man in den einzelnen Projekten durchaus von der Erfahrung anderer profitieren kann. (Experte 1)*

Quizzbreaker hingegen, wurde von allen drei befragten Experten als störend empfunden. Laut Meinung der Experten war das Projekt zu kurz, um einen Nutzen aus einem solchen Teambuilding Tool zu ziehen. Außerdem haben sich die betroffenen Projektmitglieder vorab, im Zuge des Workshops, persönlich kennen gelernt, was den Einsatz eines solchen Tools, laut Aussagen der Experten, ebenfalls unnötig macht. Experte 3 ist außerdem der Meinung, dass für das Teambuilding und bessere Kennenlernen der einzelnen Teammitglieder MS-Teams ausreichend ist.

> *Ich glaube es ist sinnvoller, zum Beispiel vor dem Standup, kurz mit den Kollegen „privat“ zu sprechen, als so ein Tool zu verwenden (…) Wir haben uns auch nie über irgendwelche Inhalte, die in den Fragen beantwortet werden mussten, unterhalten, außer dass es nervt, da waren sich alle einig. (Experte 3)*

Bezüglich der aufgestellten Kommunikationsregeln sind sich die Experten einig, dass diese aufgrund der überschaubaren Teamgröße für den Projekterfolg unwesentlich waren.

> *Solche Regeln machen, denke ich, eher Sinn für größere Teams, bei denen es ohnehin Probleme in der Kommunikation gibt. Wir haben die Regeln zwar beim initialen Training von [RAD] erläutert bekommen, es hat aber auch ohne die ganz gut funktioniert. (Experte 1)*

Experte 3 sieht ein Problem bei zu viel Reglementierung und regt an, Kommunikationsregeln bei Bedarf teamintern zu definieren.

> *Regeln sind zwar gut gemeint, aber das ist zu viel Micromanagement von außen. Wir können sie ja als Best Practices herausgeben, aber strenge Regeln sollte man vielleicht das Team selbst definieren lassen. (Experte 3)*

4.4.3 **Betrieb**

Seit dem Abschluss des Projektes, wird die Anwendung eigenständig von der Fachabteilung weiterbetreut. Das bedeutet, Änderungen werden von den Kollegen der Fachabteilung selbständig umgesetzt und getestet. Bei Problemen während der Umsetzungsphase stehen die Kollegen des RAD CoE, wie in Kapitel 4.2.4 beschrieben, jedoch weiterhin zur Verfügung. Deployments werden, wie in Kapitel 4.2.1.1 beschrieben, weiterhin zentral vom RAD Service durchgeführt. Die dadurch entstehende Abhängigkeit der Fachabteilung zum RAD Service, wird von den befragten Experten jedoch nicht als negativ eingestuft, da die Veröffentlichung, laut Aussagen der Experten, sehr zeitnah geschieht. Aufgrund der Tatsache, dass seit dem Go-Live der Anwendung nur minimale Anpassungen seitens der Fachabteilung getätigt wurden, kann die Kategorie Betrieb zu diesem Zeitpunkt von den Experten noch nicht vollständig beantwortet werden.

4.4.4 **Sonstiges / Vergangene Erfahrungen**

Experte 1 sieht beim im RAD Service umgesetzten Projekt, im Vergleich zu vergangenen, „klassischen“ Projekten, einen erhöhten Zeit- und Ressourcenaufwand, welcher sich, nach Meinung des Experten, jedoch über einen längeren Zeitraum betrachtet, rentiert:

> *Du bezahlst die Möglichkeit der Mitgestaltung schlussendlich mit Arbeitszeit und dementsprechend auch Geld. Ich denke aber, dass sich dieser Mehraufwand an Zeit und Ressourcen, über einen längeren Zeitraum betrachtet, rentiert, da die Weiterentwicklung und Anpassung der Anwendung jetzt direkt bei uns [Fachabteilung] stattfinden kann. Wir können also flexibel reagieren (…) die IT-Abhängigkeit hat sich reduziert und die Entwicklung in der Abteilung selbst ist natürlich auch günstiger. (Experte 1)*

5 DISKUSSION

Die Forschungsfrage, welche zentralen Einflussfaktoren zu berücksichtigen sind, wenn Strukturen zur modellgetriebenen Softwareentwicklung mittels Mendix in einem international verteilten Team eingeführt werden, welches über begrenztes Wissen bezüglich Softwareentwicklung verfügt, wurde, durch die Recherche und Aufbereitung einschlägiger Fachliteratur im Theorieteil dieser Arbeit, beantwortet. Dabei wurden einerseits Einflussfaktoren entlang des Software-Lebenszyklus (Kapitel 4.1.1), andererseits Einflussfaktoren, welche für eine effektive Zusammenarbeit in internationalen Teams förderlich sind (Kapitel 2.5.2), beschrieben.

Die identifizierten Einflussfaktoren können durch das im Zuge dieser Masterarbeit gegründete Center of Excellence, welches direkt im ANDRITZ RAD Servicebereich eingegliedert ist, für jedes Projekt einzeln, aufgrund von dessen Größe, Komplexität, Außenwirkung usw., beurteilt werden. Es stehen, je nach Projektanforderung, verschiedene Modelle der Zusammenarbeit zwischen IT- und Fachbereich bereit. Ebenso kann das Umsetzungsmodell an die Projektbedürfnisse angepasst werden. Das CoE dient dabei, für alle Personen im RAD Umfeld, als zentrale Anlaufstelle für jegliche, mit Low-Code Entwicklung in Verbindung stehenden, anliegen.

Das erfolgreich umgesetzte Projekt (Kapitel 4.3) einerseits sowie das die aus den Experteninterviews gewonnen Erkenntnisse (Kapitel 4.4) zeigen, dass die identifizierten Einflussfaktoren sowie das darauf aufgebaute ANDRITZ RAD CoE zur Umsetzung von Projekten in einem örtlich verteilten und interdisziplinär zusammengesetzten Team, geeignet ist. Die Hypothese H1 kann somit bestätigt, die Nullhypothese H0 verworfen werden.

Die während der Literaturrecherche identifizierten Einflussfaktoren sowie die Faktoren zur effektiven Kommunikation in verteilten Teams, konnten jedoch während der Fallstudie nicht vollständig evaluiert werden. So wird beispielsweise der Einflussfaktor „Automatisierung“, aufgrund der geringen Projektgröße sowie kurzen Projektlaufzeit, nicht in der Fallstudie berücksichtigt, da keine CI / CD Pipeline umgesetzt wurde - es können dementsprechend keine Aussagen darüber getätigt werden. Außerdem konnten aufgrund der Teamzusammensetzung (die Mitglieder waren lediglich auf Österreich, Deutschland, die Schweiz und Finnland verteilt) nicht alle Faktoren zur effektiven Zusammenarbeit in der Praxis erprobt werden. Der Faktor „Kultur“ wurde beispielsweise nicht berücksichtig.

Wie in der Fachliteratur ersichtlich und auch während den Experteninterviews bestätigt, hat der Faktor „Standards & Best Practices“ (Kapitel 4.1.1) aufgrund der Zusammensetzung des Teams (IT und Fachbereich) Einfluss auf die Produktivität bzw. Effizienz der EntwicklerInnen, speziell auf denen des Fachbereiches. Diese dokumentierten und zentral verfügbaren Richtlinien wurden, laut Aussagen der Citizen Developer / Business Developer, als hilfreich in Bezug auf das eigenständige Lösen von Projektaufgaben eingestuft und haben daher einen direkten positiven Einfluss auf den Projekterfolg. Die aufgestellte Hypothese H2 kann, anhand der Ergebnisse der Literaturrecherche als auch aufgrund der Aussagen der interviewten ExpertInnen, für die durchgeführte Fallstudie bestätigt werden.

Dem Faktor „Anforderungsanalyse“ (Kapitel 4.1.1) wird in der Literatur zugeschrieben, der häufigste Grund für das Scheitern von Projekten zu sein und ist daher von zentraler Bedeutung. Die im RAD CoE verfügbaren Methoden für Anforderungsmanagement und Projekt Setup inkl. den zur Verfügung stehenden Design Thinking Workshops, mit denen Unklarheiten in den Anforderungen beseitigt bzw. die Anforderungen bei Bedarf auch unter dem Gesichtspunkt der Usability bewertet wird, wurde von den Teilnehmern der Fallstudie in den Experteninterviews positiv bewertet.

Ebenso positiv für den Projekterfolg hat sich das in der Fachliteratur beschriebene „geteilte Führen“ (Kapitel 2.5.2) erwiesen. Im umgesetzten Projekt waren ein Product Owner (IT) und ein Business Developer (Fachbereich) zusammen für den Projekterfolg verantwortlich, was, laut den Interviews der jeweiligen Experten, als Erleichterung ihrer Arbeit im Projekt und somit positiv bewertet werden kann.

Außerdem Einfluss auf den Projekterfolgt hatte der Faktor „Koordination“ (Kapitel 2.5.2), welcher Abhängigkeiten und Wartezeiten vermeiden soll. Der gewählte Ansatz, komplexe Features, welche Abhängigkeiten aufweisen, von den Core Developern des ANDRITZ RAD CoE implementieren zu lassen wurde positiv beurteilt.

Der Faktor Kommunikationstechnologie (Kapitel 2.5.2 und 4.2.2.1) ist laut Fachliteratur in verteilten Teams das Herzstück einer effektiven Zusammenarbeit. Die im RAD CoE definierten Kanäle sind für den Zweck der erfolgreichen Projektumsetzung in einem verteilten Team, laut den Aussagen der Experten, ausreichend. Die definierten Kommunikationsregeln hingegen wurden im umgesetzten Projekt zum Teil nicht beachtet und als unnötig empfunden. Die aufgestellte Hypothese H3 kann somit nur zum Teil bestätigt werden. Herauszuheben bezüglich der verfügbaren Kommunikationsmittel ist MS-Teams, welches durchwegs positiv bewertet wurde. Yammer sowie Quizzbreaker hingegen wurden entweder wenig bis gar nicht verwendet oder gar als störend beurteilt.

Berücksichtigen sollte man jedoch auch, die bereits in Kapitel 3 erwähnten Kritikpunkte an qualitativen Forschungsmethoden, welche im Zuge dieser Arbeit zum Einsatz gekommen sind. Beispielsweise den Intervieweffekt, welcher besagt, dass meine Eigenschaften als Interviewer ein bestimmtes Antwortverhalten der interviewten Personen fördern können (beispielsweise bedingt durch ein „freundschaftliches Verhältnis“ zwischen Interviewer mit dem Experten). Durch die Tatsache, dass die beteiligten Personen bereits vor Projektbeginn wussten, dass das Projekt als Fallstudie für diese Masterarbeit dient, kann außerdem der so genannte Hawthorne-Effekt aufgetreten sein. Dieser besagt, dass es Auswirkung auf das Verhalten der Teilnehmer hat, wenn diese wissen, dass sie an einer Studie teilnehmen. Außerdem wurde lediglich eine Einzelfallstudie im Zuge dieser Masterarbeit durchgeführt und die Anzahl an Experteninterviews ist ebenfalls als gering einzuschätzen. Um die Qualität des Forschungsergebnisses zu steigern, wäre es sinnvoll, weitere Projekte, welche innerhalb des ANDRITZ RAD CoE umgesetzt werden, in ähnlicher Weise, wie in dieser Arbeit beschrieben, zu evaluieren.

Welche weiteren Schritte dazu innerhalb des ANDRITZ RAD CoE geplant sind, ist im folgenden Kapitel ersichtlich.

6 AUSBLICK

Im Zuge dieser Arbeit wurde lediglich ein einzelnes, hybrid geleitetes Projekt betrachtet, in dem außerdem ein Core Developer des RAD CoE permanent beteiligt war. Somit konnte nur ein Ausschnitt der angebotenen Leistungen des ANDRITZ RAD CoE, durch die Fallstudie und die darauffolgenden Experteninterviews, evaluiert werden. Derzeit sind innerhalb des ANDRITZ RAD Services weitere Projekte in Umsetzung, wovon zwei als kleine, nicht kritische Projekte eingestuft wurden und unter Leitung der Fachabteilung umgesetzt werden. Die restlichen in Umsetzung befindlichen Projekte werden jeweils wieder unter Leitung der IT-Abteilung durchgeführt.

Nach Abschluss dieser Projekte, wird im Zuge des kontinuierlichen Verbesserungsprozesses jeweils ein Lessons Learned Workshop abgehalten, in denen durch die von der Fachabteilung geleiteten Projekte eine neue Sichtweise auf die vom RAD CoE angebotenen Services eröffnet wird. Diese sowie die Erkenntnisse aus folgenden Projekten werden dazu beitragen, die Prozesse, Werkzeuge und Methoden des RAD CoE kontinuierlich zu optimieren und somit der Vision des ANDRITZ RAD Services, welche in Kapitel 4.2 beschrieben wurde, näher zu kommen.

Klares Ziel innerhalb der ANDRITZ Gruppe ist es, die modellgetriebene Softwareentwicklung mit Mendix als Plattform stärker zu etablieren und einem breitgefächerten Anwenderkreis auch außerhalb der IT-Abteilung zugänglich zu machen. Der derzeit eingeschlagene Weg, das zentral gesteuert vom ANDRITZ RAD CoE verschiedene Umsetzungsmethoden, Arten der Zusammenarbeit usw., je nach Projektbedarf zum Einsatz kommen, soll jedenfalls fortgesetzt werden.

ABKÜRZUNGSVERZEICHNIS

1GL	first generation Language
2GL	second generation Language
3GL	third generation Language
4GL	fourth generation Language
AD	Active Directory
API	Application Programming Interface
API	Application Programming Interface
BPML	Business Process Modeling Language
BRM	Business Relationship Manager
CASE	Computer-Aided Software Engineering
CD	Continuous Delivery
CDev	Citizen Developer
CI	Continuous Integration
SE	Systementwicklung
CSS	Cascading Stylesheets
CoE	Center of Excellence
DSL	Domain Specific Language
IDE	Integrated Development Environment
IDE	Integrated Development Environment
IT	Informationstechnik
ITSM	IT-Servicemanagement
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
OMG	Object Management Group
PDCA	Plan-do-check-act
RAD	Rapid Application Development
RegEx	Regular Expression
SDK	Software Development Kit
SME	Subject Matter Expert
QM	Qualitätsmanagement
KM	Konfigurationsmanagement
PM	Projektmanagement
SQL	Structured Query Language
SO	Sonstiges
SSO	Single Sign On
SSOT	Single Source of Truth
SVN	Subversion
UML	Unified Modeling Language

WYSIWYG	What You See Is What You Get
ECM	Enterprise Content Management

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

Acuña, S., Gómez, M., & Juristo, N. (2008). Towards understanding the relationship between team climate and software quality—a quasi-experimental study. *Empirical software engineering*, 401-434.

Albers, S., Konradt, U., Walter, A., Wolf, J., & Klapper, D. (2007). *Methodik der empirischen Forschung.* Wiesbaden: Springer Fachmedien.

ANDRITZ AG. (2020). Abgerufen am 04. 02 2020 von https://www.andritz.com/group-de/about-us

Arbeitskreis Testmanagement. (2004). Abgerufen am 21. 02 2020 von https://www.cc-gmbh.de/tavtm/seiten/testrisiken.html

Balzert, H. (2011). *Lehrbuch der Softwaretechnik.* Heidelberg: Spektrum Akad. Verl.

Berg, B., Knott, P., & Sandhaus, G. (2014). *Hybride Softwareentwicklung.* Berlin Heidelberg: Springer .

Berger, H., Beynon-Davies, P., & Cleary, P. (2004). THE UTILITY OF A RAPID APPLICATION DEVELOPMENT (RAD) APPROACH FOR A LARGE COMPLEX INFORMATION SYSTEMS DEVELOPMENT. Proceedings of the 13th European Conference on Information Systems. Abgerufen am 05. 02 2020 von https://www.researchgate.net/publication/221409923_The_Utility_of_a_Rapid_Application_Development_RAD_approach_for_a_large_complex_Information_Systems_Development

Beydeda, S., Book, M., & Gruhn, V. (2005). *Model-Driven Software Development.* Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Beynon-Davies, P., Mackay, H., & Tudhope, D. (2000). 'It's lots of bits of paper and ticks and post-it notes and things…': a case study of a rapid application development project. *Information Systems Journal*, S. 195-216.

Blueprint. (2020). Abgerufen am 09. 02 2020 von https://www.blueprintsys.com/agile-development-101/agile-and-devops

Boehm, B., & Turner, R. (2004). *Balancing agility and disciplin.* Boston: Addison-Wesley.

Bogner, A., Menz, W., & Littig, B. (2005). *Das Experteninterview - Theorie, Methode, Anwendung.* Wiesbaden: VS Verl. für Sozialwiss.

Bohnsack, R., Marotzki, W., & Meuser, M. (2011). *Hauptbegriffe qualitativer Sozialforschung.* Opladen; Farmington Hills, MI: Verlag Barbara Budrich.

Bray, I. (2002). *An introduction to requirements engineering.* Harlow: Addison-Wesley.

Britannica, E. (2020). *Encyclopædia Britannica*. Abgerufen am 15. 01 2020 von https://www.britannica.com/technology/fourth-generation-language

Bundesministerium für Digitalisierung und Wirtschaftsstandort. (2018). Abgerufen am 27. 12 2019 von Digitaldossier - Bestandsaufnahme zur Digitalisierung in Wirtschaft und Gesellschaft: https://www.bmdw.gv.at/

Dawson, B. (2016). *JAXenter*. Abgerufen am 08. 02 2020 von https://jaxenter.de/devops-nachhaltig-einfuehren-fragen-und-antworten-aus-der-praxis-40624

Dewanto, L., & Klein, M. (2018). *Heise Developer*. Abgerufen am 03. 02 2020 von https://www.heise.de/developer/artikel/Low-Code-Low-Quality-4134288.html

Dewanto, L., & Klein, M. (2018). *Heise Developer*. Abgerufen am 03. 02 2020 von https://www.heise.de/developer/artikel/Low-Code-Low-Quality-4134288.html

Ebert, C. (2014). *Systematisches Requirements Engineering.* Heidelberg : dpunkt.

Eckstein, J. (2015). *Agile Softwareentwicklung in großen Projekten.* s.l.: dpunkt.

Feldman, M., & Orlikowski, W. (2011). Theorizing practice and practicing theory. *Organization Science*, 1240-1253.

Forrester. (2019). *Low-Code Development Platforms For AD&D Professionals.* The Forrester Wave. Abgerufen am 02. 01 2020 von https://www.forrester.com/report/The+Forrester+Wave+LowCode+Development+Platforms+For+ADD+Professionals+Q1+2019/-/E-RES144387

Fowler, M. (2000). *Refactoring.* München: Addison Wesley.

Fowler, M., & Parsons, R. (2011). *Domain-specific languages.* Addison-Wesley.

Gartner. (2019). Abgerufen am 20. 12 2019 von https://www.gartner.com/doc/reprints?id=1-1ODOM46A&ct=190812&st=sb

Gartner Glossary. (2020). Abgerufen am 03. 01 2020 von Gartner Glossary: https://www.gartner.com/en/information-technology/glossary/citizen-developer

Gibson , C., & Cohen, S. (2003). *Virtual teams that work.* San Francisco: Jossey-Bass.

Gläser, J., & Laudel, G. (2010). *Experteninterviews und Qualitative Inhaltsanalyse.* Wiesbaden:: Springer.

Gumienny, R., Gericke, L., Wenzel, M., & Meinel, C. (2013). Supporting creative collaboration in globally distributed companies. *Proceedings of the 2013 conference on Computer supported cooperative work*, 995-1007.

Gurram, C., & Bandi, S. (2012). *Teamwork in Distributed Agile Software.* Sweden: School of Computing.

Hairul Nizam Nasir, M., & Sahibuddin, S. (2011). Critical success factors for software projects: A comparative study. *Scientific Research and Essays*, S. 2174-2186.

Hansen, M. (2009). *Collaboration: How leaders avoid the traps, build common ground, and recap big results.* Boston: Harvard Business Review Press.

Henkel, M., & Stirna, J. (2010). Pondering on the Key Functionality of Model Driven Development Tools: The Case of Mendix. In W. van der Aalst, J. Mylopoulos, N. M. Sadeh, M. J. Shaw, C. Szyperski, P. Forbrig, & H. Günther, *Perspectives in Business Informatics Research* (S. 146–160v). Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Hinds, P., & Bailey, D. (2003). Out of sight, out of sync: Understanding conflict in distributed teams. *Organization science*, 615-632.

Humble, J., & Farley, D. (2011). *Continuous delivery.* Upper Saddle River, NJ: Addison-Wesley.

IEEE Standard Association. (2020). Abgerufen am 19. 02 2020 von https://standards.ieee.org/standard/29148-2018.html

Industry of Things. (2017). Abgerufen am 18. 01 2020 von https://www.industry-of-things.de/warum-rapid-application-development-und-low-code-mehr-als-nur-buzzwords-sind-a-648633/

Jones, C. (1997). *Applied Software Measurement: Assuring Productivity and Quality.* New York: McGraw-Hill.

Karagiannis, D. (2013). *Enzyklopaedie der Wirtschaftsinformatik*. Abgerufen am 31. 01 2020 von https://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/is-management/Systementwicklung/Entwicklungswerkzeuge/CASE-Tools/index.html

Karis, D., Wildman, D., & Mane, A. (2014). Improving remote collaboration with video conferencing and video portals. *Human-Computer Interaction*, 1-98.

Kleppe, A., Warmer, J. B., & Bast, W. (2003). *MDA explained.* Reading, Mass.; Sebastopol, CA: Addison-Wesley; Safari Books Online.

Lamker, C. (2014). Studium und Projektarbeit (Gelbe Reihe). (T. U. Dortmund, Hrsg.) *Fallstudien*. Abgerufen am 15. 05 2020 von https://www.researchgate.net/publication/262865388_Fallstudien

Lee, K., & Zhu, X. (2017). Global virtual team performance, shared leadership, and trust: proposing a conceptual framework. *Journal of Business Economics and Management*, S. 31-38.

Leonardi, P., Treem, J., & Jackson, M. (2010). The connectivity paradox: Using technology to both decrease and increase perceptions of distance in distributed work arrangements. *Journal of Applied Communication Research*, 85-105.

Luber, S., & Litzel, N. (2019). *Big Data Insider*. Abgerufen am 15. 02 2020 von https://www.bigdata-insider.de/was-ist-data-literacy-a-823501/

Martin, J. (1982). *Application Development Without Programmers.* New Jersey: Prentice Hall.

Martin, J. (1991). *Rapid Application Development.* New York: Macmillan Pub. Co.

Maske, P. (2012). *Mobile Applikationen - Interdisziplinäre Entwicklung am Beispiel des Mobile Learning.* Wiesbaden: Springer Gabler.

Mayring, P. (2002). *Einführung in die qualitative Sozialforschung.* Weinheim und Basel: Beltz Verlag.

Mayring, P. (2010). *Qualitative Inhaltsanalyse. Grundlagen und Techniken.* Weinheim Basel: Beltz.

Medix Evaluation Guide. (2019). Abgerufen am 29. 12 2019 von https://www.mendix.com/evaluation-guide/

Mendix. (2020). Abgerufen am 02. 01 2020 von https://www.mendix.com/

Mendix Docs. (2020). Abgerufen am 02. 01 2020 von https://docs.mendix.com/

Mendix Docs. (2020). Abgerufen am 19. 02 2020 von https://docs.mendix.com/howto/general/dev-best-practices#4-2-microflows

Microsoft Docs. (2020). Abgerufen am 19. 02 2020 von https://docs.microsoft.com/de-de/microsoftteams/teams-channels-overview

Mieg, H., & Brunner, B. (2006). *Experteninterviews in den Umwelt- und Planungswissenschaften.* Lengerich: Pabst Science Publ.

Moll, K.-R., Broy, M., Pizka , M., Seifert , T., Bergner, K., & Rausch Andreas. (2004). Erfolgreiches Management von Software-Projekten. *Informatik Spektrum*, 419-432.

Nenov, A. (2018). *Medium*. Abgerufen am 04. 02 2020 von https://medium.com/@aleksandarnenov/agile-devops-and-cloud-technologies-lets-be-clear-cf986be308e1

Noack, K. (2019). *CIO von IDG*. Abgerufen am 03. 02 2020 von https://www.cio.de/a/drei-vorgehensmodelle-fuer-low-code-projekte,3545746

Object Management Group. (2014). Abgerufen am 15. 12 2019 von MDA Guide: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01

Poguntke, S. (2020). *Gabler Wirtschaftslexikon*. Abgerufen am 16. 02 2020 von https://wirtschaftslexikon.gabler.de/definition/design-thinking-54120

Quizbreaker. (2020). Abgerufen am 10. 01 2020 von https://www.quizbreaker.com/

RedHat. (2019). Abgerufen am 09. 02 2020 von https://www.redhat.com/de/topics/devops/what-is-ci-cd

Rentrop, C., Mevius, M., & Van Laak, O. (2011). Schatten-IT: ein Thema für die interne Revision? *Revisionspraxis–Journal für Revisoren, Wirtschaftsprüfer, IT-Sicherheits-und Datenschutzbeauftragte*, S. 68-76.

Revell, M. (2019). *Outsystems Blog*. Abgerufen am 29. 12 2019 von https://www.outsystems.com/blog/what-is-low-code.html

Ross, M. (2018). *InfoWorld*. Abgerufen am 03. 12 2019 von https://www.infoworld.com/article/3287146/4-essential-features-of-modern-low-code-development-platforms.html

Rouse, M. (2011). *Tech Target*. Abgerufen am 20. 02 2020 von https://searchsoftwarequality.techtarget.com/definition/Whole-team-approach

Ruparelia, N. (2010). Software development lifecycle models. *SIGSOFT Softw. Eng. Notes (ACM SIGSOFT Software Engineering Notes)*, S. 8.

Schlauch, T., Meinel, M., & Haupt, C. (2018). *Software-Engineering-Empfehlungen des DLR.*

Schulze, S., & Bo Tiedemann, J. (2018). *JAXenter*. Abgerufen am 09. 02 2020 von https://jaxenter.de/devops-wird-bizdevops-71776

Seidel, B., & Reppner , M. (2009). Business Alignment verhindert Schatten-IT. *is report*, S. 30-33.

Stahl, T., Völter, M., & Efftinge, S. (2007). *Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management.* Heidelberg: Dpunkt-Verlag.

Starke, G. (2018). *Effektive Softwarearchitekturen.* München: Hanser.

Steven, J. (2018). *Synopsys*. Abgerufen am 11. 02 2020 von https://www.synopsys.com/blogs/software-security/agile-cicd-devops-difference/

Tiemeyer, E. (2014). *Handbuch IT-Projektmanagement.* München: Hanser.

Tsun, C., & Dac-Buu, D. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, S. 961-971.

van Herpen, D., den Broeder , R., & van Ewijk , A. (2016). Verhaltensänderung als Kraftstoff für DevOps. *OBJEKTspektrum*, 1-4.

Vicent, P., Kimihiko, I., Wong, J., Yefim , N., & Driver, M. (2019). *Gartner "Magic Quadrant for Enterprise Low-Code Application Platfroms".* Gartner Research. Abgerufen am 02. 01 2020 von https://www.gartner.com/en/documents/3956079/magic-quadrant-for-enterprise-low-code-application-platf

von Kielpinski, H. (2018). *Dev Insider*. Abgerufen am 29. 01 2020 von https://www.dev-insider.de/devops-in-der-industrie-40-a-674316/

Wagner, C. (2014). *Model-Driven Software Migration.* Wiesbaden: Springer.

Weinmeister, P. (2015). *Practical Salesforce.com Development Without Code.* Apress.

Wieczorrek, H. W., & Mertens, P. (2011). *Management von IT-Projekten.* Berlin Heidelberg: Springer-Verlag.

Yin, R. K. (2009). *Case study research.* Los Angeles: SAGE.