Die digitale Transformation verändert grundlegend, wie produzierende Unternehmen IT-Lösungen nutzen, um Prozesse zu optimieren und wettbewerbsfähig zu bleiben. Gleichzeitig steigt die Nachfrage nach digitalen Anwendungen, doch der IT-Fachkräftemangel bremst deren Umsetzung. Viele Unternehmen sind stark von ihren IT-Abteilungen abhängig, die unter wachsendem Entwicklungsdruck stehen – mit der Folge, dass digitale Innovationen ins Stocken geraten.
Low-Code-Plattformen versprechen Abhilfe, indem sie auch Nicht-IT-Fachkräften ermöglichen, aktiv an der Softwareentwicklung mitzuwirken. Fachabteilungen können so eigenständig digitale Anwendungen erstellen und Prozesse effizienter gestalten. Dennoch bleibt der Einsatz von Low-Code oft auf IT-Abteilungen beschränkt – nicht zuletzt aufgrund von Bedenken hinsichtlich Datensicherheit, Governance und fehlendem Wissen über die konkreten Einsatzmöglichkeiten dieser Technologie.
Diese Dissertation entwickelt ein umfassendes Regelwerk, das produzierende Unternehmen dabei unterstützt, Low-Code gezielt, sicher und effizient zu nutzen. Drei definierte Anwendungsfalltypen schaffen ein besseres Verständnis für die Einsatzmöglichkeiten. Zudem werden elf grundlegende Regeln abgeleitet, die eine strukturierte und sichere Einführung von Low-Code-Plattformen in produzierenden Unternehmen ermöglichen. Durch die Analyse der Zusammenhänge zwischen Anwendungsfällen und Regeln lassen sich spezifische Maßnahmen ableiten, um Low-Code sicher und effektiv einzusetzen. Ergänzend dazu werden 60 praxisnahe Gestaltungsmaßnahmen formuliert, die Unternehmen konkrete Unterstützung bei der erfolgreichen Integration von Low-Code in ihre Prozesse bieten.
Dieses Regelwerk hilft produzierenden Unternehmen, die Potenziale von Low-Code voll auszuschöpfen, Abhängigkeiten von der IT zu reduzieren und die digitale Transformation gezielt voranzutreiben.

ISBN 978-3-98555-276-4

fir an der RWTH Aachen

194

Regelwerk für Low-Code-Anwendungen in produzierenden Unternehmen

Kira Frings

Schriftenreihe Rationalisierung

Kira Frings

Regelwerk für Low-Code-Anwendungen in produzierenden Unternehmen

fir an der RWTH Aachen

Herausgeber:
Univ.-Prof. Dr.-Ing. Dipl.-Wirt. Ing. Günther Schuh

194

Edition Wissenschaft Apprimus

Regelwerk für Low-Code-Anwendungen in produzierenden Unternehmen

Rule Set for Low-Code Applications in Manufacturing Companies

Von der Fakultät für Maschinenwesen
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades einer
Doktorin der Ingenieurwissenschaften
genehmigte Dissertation

vorgelegt von

Kira Julia Frings

Berichter/in:

Univ.-Prof. Dr.-Ing. Dipl.-Wirt. Ing. Günther Schuh
apl. Prof. Dr.-Ing. Wolfgang Boos

Tag der mündlichen Prüfung: 04. März 2025

SCHRIFTENREIHE RATIONALISIERUNG

Kira Frings

Regelwerk für Low-Code-Anwendungen in produzierenden Unternehmen

Herausgeber:
Prof. Dr.-Ing. Dipl.-Wirt. Ing. G. Schuh

Band 194

fir an der RWTH Aachen

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über https://portal.dnb.de abrufbar.

Kira Frings:

Regelwerk für Low-Code-Anwendungen in produzierenden Unternehmen

1. Auflage, 2025

Gedruckt auf holz- und säurefreiem Papier, 100% chlorfrei gebleicht.

Wissenschaftsverlag des Instituts für Industriekommunikation und Fachmedien
an der RWTH Aachen
Steinbachstr. 25, 52074 Aachen, Deutschland
Internet: www.apprimus-verlag.de, E-Mail: info@apprimus-verlag.de

Printed in Germany

ISBN 978-3-98555-276-4

D 82 (Diss. RWTH Aachen University, 2025)

Danksagung

Die vorliegende Dissertation ist das Ergebnis einer intensiven und bereichernden Zeit am FIR an der RWTH Aachen. Ohne die Unterstützung und Begleitung vieler Menschen wäre diese Arbeit nicht möglich gewesen. Ihnen allen gilt mein tief empfundener Dank.

Ein großes Dankeschön geht an meine Kolleginnen und Kollegen am FIR – für den inspirierenden Austausch, die gemeinsame Arbeit an spannenden Projekten und die vielen wertvollen Diskussionen. Besonders danken möchte ich all jenen, die mich auf diesem Weg begleitet und unterstützt haben. Viele von ihnen waren nicht nur wertvolle Wegbegleiter, sondern auch Leidensgenossen auf dem steinigen Weg zur Promotion – gemeinsam haben wir Herausforderungen gemeistert, Rückschläge überwunden und Erfolge gefeiert. Hierbei möchte ich insbesondere Murtaza Abbas, Yannick Becerra, Justus Benning, Sijmen Boersma, Sebastian Bredel, Robin Cen, Florian Clemens, Dr. Jacques Engländer, Dr. Markus Fischer, Lars Frehen, Clara Herkenrath, Dr. Jan Hicking, Gerrit Hoeborn, Dr. Jörg Hoffmann, Sebastian Junglas, Cansu Kahveci, Kajan Kandiah, Abiraam Kantharajah, Gurleen Kaur, Sebastian Kremer, Stefan Leachu, Kerstin Lörsch, Justus Mund, Dr. Jonas Müller, Jessica Rahn, Jonas Reck, Regina Schrank, Tobias Schröer, Fabian Seidel, Stefan Sarawinsky, Max-Ferdinand Stroh, Dr. Lukas Stratmann, John von Stamm, Tim Walter, David Waßmann und Franziska Zielenbach hervorheben.

Ebenso danke ich meinen Freundinnen und Freunden, die mich in den vergangenen Jahren begleitet haben – sei es durch fachlichen Austausch, motivierende Gespräche oder einfach durch die richtige Ablenkung zum richtigen Zeitpunkt. Besonders hervorheben möchte ich meine Mitbewohnerinnen Sophia Ossig und Tessa Frings, ohne deren Unterstützung ich mich nicht so intensiv auf die Dissertation hätte fokussieren können.

Mein tiefster Dank gilt meiner Familie, besonders meiner Mutter. Ihr habt mich stets unterstützt, mir den Rücken freigehalten und mich motiviert, wenn es schwierig wurde. Ohne euch wäre dieser Weg nicht möglich gewesen.

Diese Arbeit ist nicht nur mein eigenes Werk, sondern das Ergebnis vieler Menschen, die mich inspiriert, herausgefordert und unterstützt haben. Danke!

Köln, 13.03.2025 Kira Frings

Zusammenfassung

Die digitale Transformation revolutioniert branchenübergreifend, wie Unternehmen mithilfe digitaler Technologien Mehrwerte schaffen. Angesichts eines steigenden Bedarfs an digitalen Lösungen bei gleichzeitigem Fachkräftemangel im IT-Bereich stehen produzierende Unternehmen vor der Herausforderung, dieser Nachfrage gerecht zu werden. Low-Code-Plattformen bieten eine Lösung, indem sie Nicht-IT-Fachkräften ermöglichen, am Entwicklungsprozess teilzunehmen und diesen so beschleunigen und vereinfachen. Trotz des großen Potenzials von Low-Code bleibt die Softwareentwicklung jedoch meist in den Händen der IT-Abteilungen. Bedenken hinsichtlich der Datensicherheit und des Datenschutzes zählen zu den Hauptgründen gegen den Einsatz von Low-Code, was auf ein mangelndes Verständnis von Low-Code-Anwendungsmöglichkeiten und fehlender Low-Code-Governance zurückzuführen ist.

Das Ziel dieser Dissertation ist die Entwicklung eines Regelwerks für Low-Code-Anwendungen, das spezifische Regeln und Maßnahmen für produzierende Unternehmen umfasst. Hierfür wurden vier Modelle entwickelt:

1. Definition von drei Low-Code-Anwendungsfalltypen zur Schaffung eines Verständnisses für die Einsatzmöglichkeiten von Low-Code.
2. Entwicklung von elf Low-Code-Regeln, aufgeteilt in fünf Dimensionen, um ein Fundament für die Implementierung und den Betrieb von Low-Code-Anwendungen zu etablieren.
3. Untersuchung der Wirkbeziehungen zwischen den Low-Code-Anwendungsfalltypen und Regeln durch Expert*inneninterviews, um anwendungsspezifische Maßnahmen zur Einhaltung der Low-Code-Regeln abzuleiten.
4. Formulierung von 60 Gestaltungsmaßnahmen und einem Vorgehen zur Anwendung des Regelwerks, um Unternehmen eine zielgerichtete Unterstützung bei der Implementierung von Low-Code zu bieten.

Zusammengefasst liefert diese Dissertation eine umfassende Unterstützung für den Einsatz von Low-Code-Anwendungen in produzierenden Unternehmen, indem sie ein klares Verständnis der unterschiedlichen Low-Code-Anwendungsfälle, damit verbundener Risiken und erforderlicher Governance-Strukturen bietet. Durch die systematische Entwicklung des Low-Code-Regelwerks und die Ableitung praxisorientierter Gestaltungsmaßnahmen wird ein wichtiger Beitrag zur Überbrückung der Lücke zwischen technologischen Möglichkeiten und deren effektiver Nutzung in der Praxis geleistet.

Summary

Digital transformation is revolutionizing how companies across industries create value through digital technologies. Faced with a rising demand for digital solutions amidst a simultaneous shortage of IT specialists, manufacturing companies are challenged to meet this demand. Low-Code platforms offer a solution by enabling non-IT professionals to participate in the development process, thereby accelerating and simplifying it. Despite the obvious advantages and significant potential of Low-Code, software development often remains primarily in the hands of IT departments. Concerns about data security and data protection are among the main reasons against using Low-Code, often due to a lack of understanding of Low-Code's application possibilities and missing IT governance.

The goal of this dissertation is to develop a set of rules for Low-Code applications that includes specific rules and measures for manufacturing companies. To this end, four models were developed:

1. Definition of three types of Low-Code use cases to foster an understanding of the possibilities of using Low-Code.
2. Development of eleven Low-Code rules divided into five domains to establish a foundation for the control and monitoring of Low-Code applications.
3. Examination of the relationships between the types of Low-Code use cases and the rules through expert interviews to derive use case-specific measures for complying with the Low-Code rules.
4. Formulation of 60 design measures and a procedure for applying the set of rules in order to offer companies targeted support in implementing Low-Code.

In summary, this dissertation offers a comprehensive support for integrating Low-Code into manufacturing companies by providing a clear understanding of the various Low-Code application cases, associated risks, and necessary governance structures. Through the systematic development of the framework and the derivation of practice-oriented design recommendations, it makes a significant contribution to bridging the gap between technological possibilities and their effective use in practice.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

EAM	**E**nterprise **A**rchitecture **M**anagement
FIR	**F**orschungs**i**nstitut für **R**ationalisierung
GRC	**G**overnance-**R**isk-**C**ompliance
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
MDSD	**M**odel-**D**riven **S**oftware **D**evelopment
MES	**M**anufacturing-**E**xecution-**S**ystem
RAD	**R**apid **A**pplication **D**evelopment
SDLC	Softwareentwicklungs-Lebenszyklus (**S**oftware **D**evelopment **L**ife **C**ycle)
SLA	**S**ervice-**L**evel-**A**greement
UI	**U**ser **I**nterface

1 Einleitung

1.1 Ausgangssituation und Problemstellung

Die digitale Transformation durchdringt alle Branchen und definiert neu, wie Unternehmen durch den Einsatz digitaler Technologien Werte schaffen (s. HESS ET AL. 2016). Digitale Technologien und die damit verbundenen Industrie 4.0-Konzepte sind für produzierende Unternehmen essenziell, um wettbewerbsfähig zu bleiben (s. SCHUH ET AL. 2020). Produzierende Unternehmen stehen bei dem Einsatz von digitalen Technologien allerdings vor der Herausforderung, die steigende Nachfrage nach digitalen Lösungen zu erfüllen (s. HOOGSTEEN U. BORGMAN 2022), da ein internationaler Mangel an IT-Fachkräften, insbesondere im Bereich der Softwareentwicklung, herrscht (s. NITHITHANATCHINNAPAT U. JOSHI 2019). Die steigende Nachfrage und der gleichzeitige Mangel an qualifizierten IT-Ressourcen, führt zu Engpässen und einer zunehmenden Diskrepanz zwischen IT-Nachfrage und -Angebot (s. LEBENS ET AL. 2022). Gleichzeitig führt die fortschreitende Digitalisierung zu spezifischeren Problemen, wodurch das Einbeziehen der Nutzer von digitalen Technologien zunehmend wichtiger wird (s. URBACH U. AHLEMANN 2019). Wenn IT-Abteilungen der IT-Nachfrage der Nutzer nicht gerecht werden können, greifen Fachabteilungen zunehmend auf sogenannte Schatten-IT zurück (s. KNOLL U. STRAHRINGER 2017). Obwohl diese Schattenanwendungen anfangs die Produktivität steigern, werden die Anwendungen nicht von der IT-Abteilung unterstützt und können zu IT-Sicherheitsproblemen und einem Mangel an Transparenz führen (s. HEUER ET AL. 2022).

Als Lösungsansatz kann die IT-Abteilung ihre Organisation agiler gestalten und die IT-Architektur im Unternehmen modernisieren (s. KNOLL U. STRAHRINGER 2017). Low-Code-Plattformen, mit einer grafischen Oberfläche und vorgefertigten Code-Bausteinen, ermöglichen es auch Nicht-IT-Spezialist*innen, digitale Lösungen zu entwickeln (siehe Kapitel 2.1.2) und somit zur digitalen Transformation ihrer Organisationen beizutragen (s. IHO ET AL. 2021). Dies ermöglicht einen schnelleren und effizienteren Entwicklungsprozess, da die Anforderungen nicht vollständig zwischen den Fachbereichen und der IT abgestimmt und übersetzt werden müssen (s. CARROLL ET AL. 2022). IT-Abteilungen können so entlastet werden. Abbildung 1-1 zeigt die drei Hauptgründe für die Verwendung von Low-Code-Plattformen.

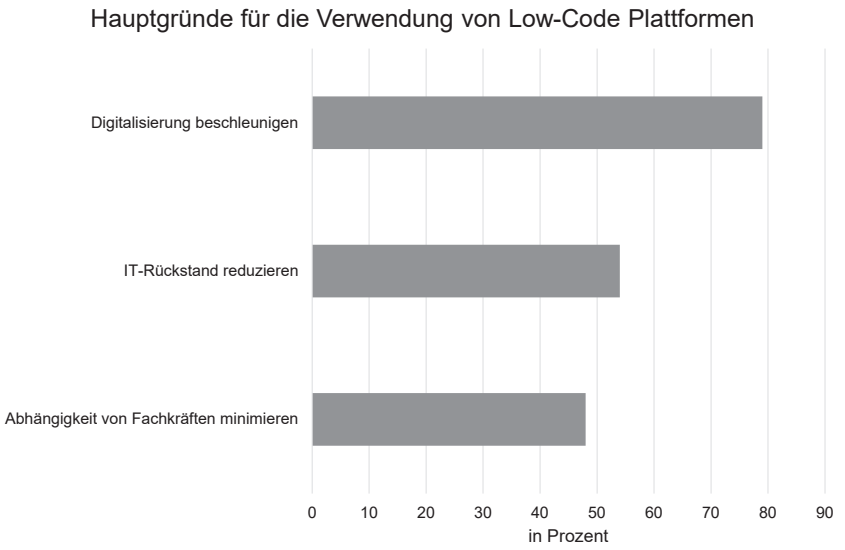

Abbildung 1-1: Hauptgründe für die Verwendung von Low-Code-Plattformen (i. A. a. JUHAS ET AL. 2022)

Durch Low-Code-Plattformen kann die Abhängigkeit von IT-Fachkräften minimiert und der IT-Rückstand reduziert werden, was zu einer beschleunigten Digitalisierung des Unternehmens führen kann (s. JUHAS ET AL. 2022). Trotz dieser Vorteile und des erheblichen Potenzials von Low-Code-Plattformen liegt die Softwareentwicklung in vielen Unternehmen weiterhin vorrangig in den Händen der IT-Abteilungen (s. TALESRA U. G. S. 2021). Oft besteht bei den Fachbereichen Zurückhaltung, digitale Lösungen eigenständig zu entwickeln, selbst wenn ein signifikanter Rückstau an weniger priorisierten IT-Projekten existiert (s. TECHCONSULT GMBH 2021). Datensicherheit und Datenschutz werden dabei als einer der Hauptgründe genannt, derzeit keine Low-Code-Plattformen einzusetzen oder deren Einsatz auch in Zukunft auszuschließen (s. TECHCONSULT GMBH 2021). Darüber hinaus stellt die richtige Auswahl der Low-Code-Plattformen eine Herausforderung dar. Unterschiedliche Low-Code-Plattformen haben verschiedene Beschränkungen in Bezug auf die Anpassungsfähigkeit und eignen sich unterschiedlich gut für die Entwicklung komplexer Anwendungen (s. JUHAS ET AL. 2022). Da der Funktionsumfang der Low-Code-Plattformen stark variiert, eignet sich je nach Art der Low-Code-Anwendung eine unterschiedliche Low-Code-Plattform (s. SCHAFFRY 2023).

In der Praxis zeigt sich, dass Geschäft und IT ihre jeweilige Rolle in Bezug auf den Einsatz von verschiedenen Low-Code-Anwendungen noch finden müssen und dass sie daher ihre Zusammenarbeitsmentalität ändern müssen (s. PRINZ ET AL. 2022). Dabei ist es von entscheidender Bedeutung, dass sich die Low-Code-Entwickler*innen aus den Fachbereichen der von der IT festgelegten Richtlinien und Standards bewusst

sind (s. BINZER U. WINKLER 2022). Das Fehlen von Governance- und Überwachungsmechanismen beim Einsatz von Low-Code führt zu einem unkontrollierten Einsatz von Datenschnittstellen und zu einem Mangel an Kontrolle über die Anwendungsentwicklung und -bereitstellung (s. HEUER ET AL. 2022). Diese Situation birgt Risiken in Bezug auf Datensicherheit, Compliance und das Potenzial für die Schaffung von Schatten-IT-Umgebungen (s. HEUER ET AL. 2022).

Um die Nutzung von Low-Code-Plattformen effektiv zu steuern und zu überwachen, bedarf es einer starken IT-Governance Struktur, um sicherzustellen, dass entwickelte Anwendungen den Unternehmensstandards entsprechen und Sicherheitsrisiken minimiert werden (s. SANCHIS ET AL. 2020). Eine Herausforderung bei der Nutzung von Low-Code-Plattformen ist es, die Konsistenz und Qualität von Anwendungen zu gewährleisten. Insbesondere dann, wenn Low-Code-Entwickler*innen ohne tiefe technische Kenntnisse involviert sind (s. KHORRAM ET AL. 2020). Es müssen Regeln und Maßnahmen bereitgestellt werden, die eine qualitativ hochwertige und sichere Anwendungsentwicklung fördern, während gleichzeitig die Flexibilität und Agilität, die Low-Code-Plattformen bieten, erhalten bleiben (s. KHORRAM ET AL. 2020). Dies erfordert eine Anpassung der bestehenden IT-Governance mit gut definierten Entscheidungsprozessen, Rollen und Verantwortlichkeiten sowie relationalen Mechanismen für die Implementierung und den Betrieb von Low-Code (s. PRINZ ET AL. 2022). Die Wissenschaft weist jedoch auf aktuelle Forschungslücken hinsichtlich der IT-Governance von Low-Code-Plattformen hin (s. PRINZ ET AL. 2022).

Um die bestehenden Forschungslücken zu schließen und praktischen Nutzen zu generieren, ist es notwendig, dass die Autorin dieser Dissertation den Einfluss unterschiedlicher Low-Code-Anwendungsfälle auf Unternehmen sowohl technisch als auch organisatorisch beleuchtet. Dazu wird untersucht, welche Low-Code-Anwendungsfälle in der Praxis in Unternehmen vorkommen. Anschließend werden die Risiken beim Einsatz von Low-Code beschrieben und in Form von Regeln zur Risikovermeidung festgehalten. Im nächsten Schritt wird die Wirkbeziehung zwischen den verschiedenen Low-Code-Anwendungsfällen und den Low-Code-Regeln erklärt. Die gewonnenen Erkenntnisse werden anschließend in ein strukturiertes Regelwerk überführt, das neben den zuvor identifizierten Regeln auch Gestaltungsmaßnahmen enthält, die beschreiben, wie die Regeln anwendungsspezifisch eingehalten werden können. Durch dieses Vorgehen wird untersucht, wie sich die Implementierung und der Einsatz von Low-Code-Anwendungen auf die IT-Architektur sowie auf die Organisationsstruktur von Unternehmen auswirken. Das Regelwerk soll dazu dienen, die bestehende IT-Governance von produzierenden Unternehmen in Bezug auf Low-Code zu erweitern.

1.2 Zielsetzung und Forschungsfrage

Im Kontext der zuvor beschriebenen Ausgangslage und der identifizierten Problemstellung verfolgt diese Dissertation das Ziel, ein Regelwerk für Low-Code-Anwendung in produzierenden Unternehmen zu entwickeln. Dieses Regelwerk soll präzise Regeln und unterstützende Maßnahmen für spezifische Anwendungsfälle bereitstellen und als

Erweiterung der bestehenden IT-Governance verwendet werden. Es wird ein systematischer Ansatz vorgeschlagen, der für verschiedene Low-Code-Anwendungsfalltypen spezifische Low-Code-Regeln liefert. Auf dieser Grundlage können anwendungsfallspezifische Maßnahmen abgeleitet werden. Daraus ergibt sich die zentrale Forschungsfrage:

Welche Regeln sollte ein produzierendes Unternehmen für Low-Code-Anwendung mindestens festlegen?

Diese Hauptforschungsfrage wird durch vier detaillierte und aktionsorientierte Unterfragen ergänzt und vertieft. Die Zielsetzung, Hauptforschungsfrage sowie die vier Unterfragen sind in Abbildung 1-2 dargestellt.

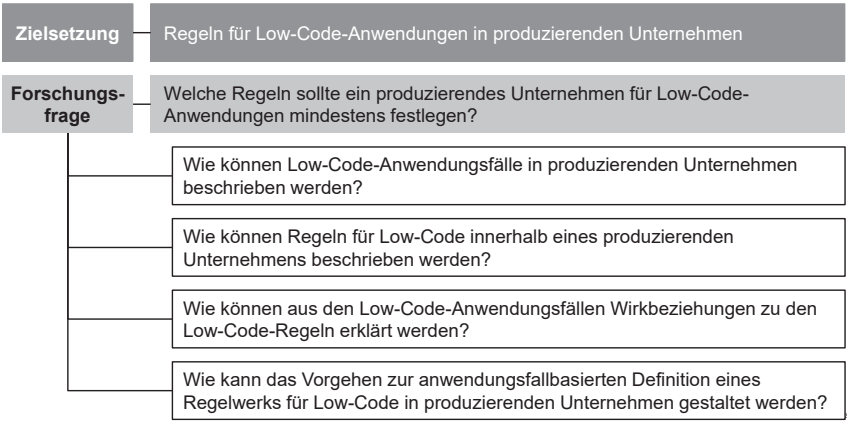

Abbildung 1-2: Zielsetzung und Forschungsfragen der vorliegenden Arbeit (eigene Darstellung)

Das erste Teilziel zur Erreichung des Forschungsziels besteht in der systematischen Analyse, Identifikation und Strukturierung von Low-Code-Anwendungsfällen. Die technologische Entwicklung im Bereich Low-Code gewinnt zunehmend an Bedeutung für die digitale Transformation produzierender Unternehmen. Diese Entwicklung zeichnet sich durch ein breites Spektrum an Low-Code-Anwendung aus. Es existieren keine Typisierungsansätze für die verschiedenen Arten von Low-Code-Anwendung. Aufgrund der schnellen Entwicklungsfortschritte fehlt ein klares Verständnis dafür, wie verschiedene Anwendungsmerkmale die Organisationsstruktur und die IT-Architektur eines Unternehmens beeinflussen. Deshalb ist die Entwicklung eines konsistenten Verständnisses für die verschiedenen Arten von Low-Code-Anwendungsfällen erforderlich.

Das zweite Teilziel konzentriert sich darauf, relevante Regeln für die Implementierung und den Betrieb von Low-Code-Anwendung zu präzisieren. Durch die Integration von bestehenden IT-Governance Rahmenwerken können Strukturierungsdimensionen für das Low-Code-Regelwerk identifiziert werden. Basierend auf den in der Literatur

identifizierten Risiken und Herausforderungen von Low-Code können relevante Low-Code-Regeln für produzierende Unternehmen abgeleitet werden.

Das dritte Teilziel befasst sich mit der Untersuchung der gegenseitigen Einflüsse zwischen den verschiedenen Typen von Low-Code-Anwendungsfällen und den Low-Code-Regeln. Entsprechend einem praxisnahen Forschungsansatz sollen die erzielten Erkenntnisse genutzt werden, um geeignete Maßnahmen zur Adressierung der identifizierten Risiken und Herausforderungen zu entwickeln.

Das vierte Teilziel strebt danach aus den zuvor identifizierten Maßnahmen konkrete Gestaltungsempfehlungen zu formulieren. Dazu werden sowohl typenunabhängige als auch anwendungsfalltypspezifische Gestaltungsemfehlungen auf Basis der Maßnahmen präsentiert.

Zusammengefasst zielt die in dieser Arbeit entwickelte systematische Methode darauf ab, IT- und Fachbereiche zu befähigen, die Risiken der Komplexität von Low-Code-Anwendungsfällen und deren Auswirkungen auf die Organisationsstruktur und IT-Architektur mittels spezifischer Gestaltungsempfehlungen effektiv zu bewerten und gezielt zu managen. Das vorgeschlagene Regelwerk unterstützt dabei, anwendungsfallspezifische Maßnahmen zu ergreifen, um eine nachhaltige Implementierung und den Betrieb von Low-Code-Anwendung in produzierenden Unternehmen sicherzustellen.

1.3 Wissenschaftlicher Bezugsrahmen und Forschungskonzeption

Im Anschluss an die Darlegung der Ziele wird nun der wissenschaftliche Bezugsrahmen dieser Arbeit erörtert, wofür ein Verständnis der Wissenschaftstheorie, auch als „Wissenschaft der Wissenschaften“ (s. ULRICH U. HILL 1976, S. 305) bezeichnet, erforderlich ist. Abbildung 1-3 zeigt die Aufteilung der Wissenschaften nach ihren Zielen und bietet Beispiele für jede Kategorie.

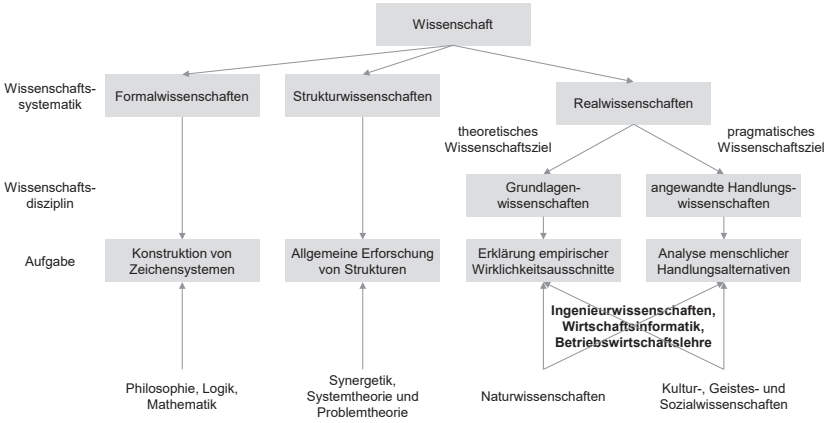

Abbildung 1-3: Wissenschaftssystematik (i. A. a. ULRICH U. HILL 1976, S. 305; ZELEWSKI 2008, S. 3; HOFFMANN 2018, S. 6)

Die Formalwissenschaften, wie zum Beispiel die Mathematik, beschäftigen sich mit der Konstruktion von wissenschaftlichen Sprachsystemen, von Sprachen und Zeichen, wie zum Beispiel die Mathematik, und verfolgen dabei eine logische Vorgehensweise (s. ULRICH U. HILL 1976, S. 305). Strukturwissenschaften, wie beispielsweise die Systemtheorie oder die Synergetik, befassen sich mit Strukturen von realen und formalen Objekten (s. ZELEWSKI 2008, S. 3). Die Realwissenschaften untersuchen dagegen real existierende Objekte und Vorkommnisse, wie es zum Beispiel die Naturwissenschaften, aber auch Geistes- und Sozialwissenschaften sowie die Wirtschaftswissenschaften tun (s. JUNG 2012, S. 22).

Die Realwissenschaften können weiter in Grundlagenwissenschaften und angewandte Handlungswissenschaften unterteilt werden. Bei den Grundlagenwissenschaften steht das Ziel der Erklärung von real existierenden Gesetzmäßigkeiten im Vordergrund (s. ZELEWSKI 2008, S. 5). Dagegen verfolgen die angewandten Handlungswissenschaften ein eher pragmatisches Wissenschaftsziel – die Anwendbarkeit der Modelle steht vor der Allgemeingültigkeit von Theorien (ULRICH 1984, S. 174s. ; LEHNER ET AL. 2008, S. 20; ÖSTERLE ET AL. 2010, S. 665). Sie beschäftigen sich daher mit der Gestaltung künftiger Realitäten, während sich die Grundlagenwissenschaften mit der Erklärung existierender Realitäten befassen (s. ULRICH 1984, S. 179).

Die vorliegende Arbeit befasst sich mit Fragestellungen der Ingenieurwissenschaften, der Betriebswirtschaftslehre und der Wirtschaftsinformatik. Ingenieurwissenschaften sind wegen der Untersuchung technisch orientierter Objekte genauso den angewandten Handlungswissenschaften zuzuordnen wie die Betriebswirtschaftslehre, die Modelle bildet, um anwendungsorientierte Handlungsempfehlungen zu geben (s. ULRICH U. HILL 1976, S. 305; WÖHE ET AL. 2016, S. 12). Die Wirtschaftsinformatik hat zwar einen strukturwissenschaftlichen Charakter, kann aber wegen ihres Anwendungsbezugs auf ein sachliches Objektsystem auch als Ingenieurwissenschaft eingeordnet werden (s. FINK ET AL. 2005, S. 256). Die in der vorliegenden Arbeit behandelten Themen der Low-Code-Anwendungsfälle und Regeln sind aus der Praxis hergeleitet und es wird die künftige Gestaltung eines Low-Code-Regelwerks erforscht. Damit und mit der Einordnung der grundsätzlichen Fragestellungen kann die vorliegende Arbeit den angewandten Handlungswissenschaften zugeordnet werden.

In den Realwissenschaften müssen grundsätzlich die Probleme der Subjektivität und der Kommunikation angegangen werden. Subjektivität bezieht sich auf die persönlichen Wahrnehmungen der Forschenden, die durch individuelle Erfahrungen und gewohnte Interpretationsmuster geprägt sind. Das Problem der Kommunikation hingegen betrifft die Herausforderung, komplexe Sachverhalte präzise und verständlich zu vermitteln, was insbesondere in abstrakten, jedoch praxisorientierten Forschungsarbeiten von großer Bedeutung ist (s. ULRICH U. HILL 1976, S. 306). Angesichts dieser Problematiken lässt sich, basierend auf der Einordnung dieser Arbeit, ein geeigneter Forschungsprozess ableiten, der auf die Überwindung dieser Herausforderungen ausgerichtet ist (siehe Abbildung 1-4).

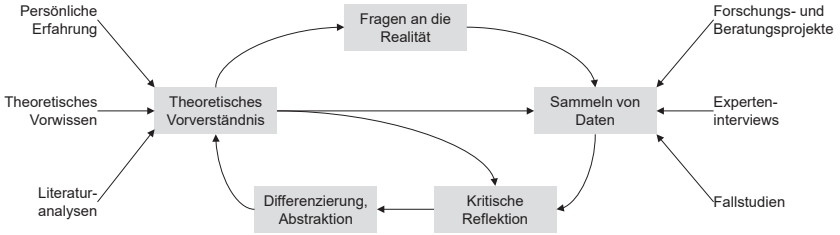

Abbildung 1-4: Forschungskonzeption (eigene Darstellung i. A. a. KUBICEK 1977, S. 14-15)

Der Forschungsprozess basiert auf dem Prinzip des zunehmenden Verständnisses der Realität nach KUBICEK. Dieser Ansatz, der ein exploratives Vorgehen favorisiert, ermöglicht es, auf der Grundlage eines vorhandenen Vorverständnisses im Forschungsfeld sowie der Erfahrungen von Expert*innen, gezielt Fragen zu stellen und auf Basis eigener Hypothesen neue Erkenntnisse zu generieren (s. EISEND U. KUß 2017, S. 12-13; KUBICEK 1977, S. 14-15).

Um dem Problem der Subjektivität zu begegnen, ist das Vorverständnis der Autorin entscheidend (s. MAYRING 2002, S. 25). Dieses ist durch deren berufliche Erfahrung unter anderem durch die Mitarbeit und Leitung von industriellen Beratungsprojekten in produzierenden Unternehmen sowie durch die Bearbeitung von praxisnahen Forschungsprojekten am Forschungsinstitut für Rationalisierung e. V. an der RWTH Aachen (FIR) geprägt. Dem in dieser Arbeit adressierten Kommunikationsproblem wird durch die Bereitstellung umfassender Definitionen im zweiten Kapitel begegnet. Präzise definierte Begrifflichkeiten sind entscheidend, um die Verlässlichkeit theoretischer Aussagen zu gewährleisten (s. EISEND U. KUß 2017, S. 14). Hierbei wird, soweit möglich, auf bereits etablierte Definitionen im Bereich des Informationsmanagements zurückgegriffen und diese werden bei Bedarf um praxisrelevante Termini erweitert.

Die sich aus dem dargelegten Forschungsprozess ergebenden Aufgaben lassen sich nach ULRICH U. HILL in terminologisch-deskriptive, empirisch-induktive und analytisch-deduktive Aufgaben aufteilen. Zu den terminologisch-deskriptiven Aufgaben zählen die Definition von Begriffen und deren Anwendung sowie die Abgrenzung des Forschungsbereichs durch Festlegung relevanter Dimensionen und die Bildung von Typologien. Im Bereich der empirisch-induktiven Aufgaben erfolgt die Hypothesengenerierung basierend auf der Beobachtung von Zusammenhängen sowie die empirische Überprüfung dieser Hypothesen. Analytisch-deduktive Aufgaben umfassen schließlich logische Prozesse, wie die Modellentwicklung und die Ableitung von situationsspezifischen Handlungsalternativen (s. ULRICH U. HILL 1976, S. 348).

Zur Forschungskonzeption dieser Arbeit werden neben den bereits genannten Methoden auch Verfahren der Sozialforschung hinzugefügt. Hierzu gehören strukturierte Literaturrecherchen sowie Expert*inneninterviews, um das Forschungsvorgehen kontinuierlich zu überprüfen und weiterzuentwickeln (s. DÜNNEBACKE 2015, S. 13).

1.4 Aufbau der Arbeit

Die Dissertation dokumentiert den Forschungsprozess in einer strukturierten Form und ist in zehn Kapitel unterteilt. Sie orientiert sich an den Phasen der angewandten Forschung gemäß ULRICH und wird in Abbildung 1-5 visualisiert (s. ULRICH 1984, S. 20).

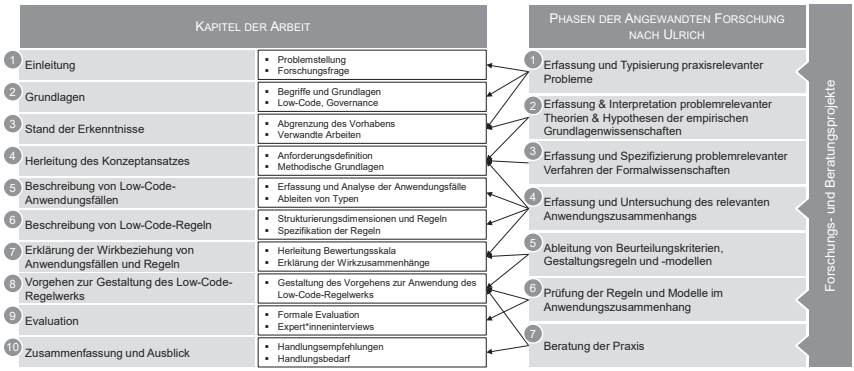

Abbildung 1-5: Phasen der angewandten Forschung nach ULRICH u. HILL (eigene Darstellung i. A. a. HOFFMANN 2018, S. 10)

Die Einleitung legt den Grundstein dieser Arbeit und wird gefolgt von Kapitel 2, das sich der Erläuterung der für das Verständnis der Leserschaft wichtigen Grundlagen widmet. Hierzu zählen die Definition zentraler Begriffe sowie die Abgrenzung des Untersuchungsbereichs der Arbeit. Kapitel 3 deckt auf Basis strukturierter Literaturrecherchen den Forschungsbedarf auf, der sich aus der Sichtung aktueller deutschsprachiger und internationaler Literatur ergibt. Es werden zudem relevante Vorarbeiten anderer Autor*innen vorgestellt, die in dieser Arbeit aufgegriffen und weitergeführt werden. Kapitel 4 etabliert die methodischen Grundpfeiler für die im Verlauf der Arbeit entwickelten Modelle, legt die Anforderungen fest und beschreibt das methodische Vorgehen.

Im Mittelpunkt der Arbeit stehen die zwei Beschreibungsmodelle. Das erste Beschreibungsmodell in Kapitel 5 konkretisiert zunächst den Begriff der Low-Code-Anwendungsfälle anhand konstituierender Merkmale. Die detaillierenden Merkmale von Low-Code-Anwendungsfällen werden im Anschluss mit ihren Ausprägungen vorgestellt. Die Typisierung der Anwendungsfälle erfolgt anhand der logischen und verifizierbaren Verknüpfung der zuvor vorgestellten Merkmalsausprägungen. Das zweite Beschreibungsmodell in Kapitel 6 widmet sich der Darstellung von Low-Code-Regeln, strukturiert nach dem IT-Governance COBIT-Kernmodell. Durch eine strukturierte Inhaltsanalyse werden einzelne Low-Code-Regeln abgeleitet und detailliert beschrieben.

Das Erklärungsmodell in Kapitel 7 verwendet die Low-Code-Anwendungsfalltypen und Regeln, um anwendungsfallspezifische Maßnahmen für die Einhaltung der Low-Code-Regeln zu identifizieren. Kapitel 8 beschäftigt sich mit der praktischen Anwendung der Modelle zur Gestaltung eines Low-Code-Regelwerks für produzierende Unternehmen.

Eine begrenzt empirisch-induktive Verifikation der Ergebnisse durch die exemplarische Anwendung der Modelle in Fallstudien wird in Kapitel 9 vorgestellt.

Den Abschluss der Arbeit bildet Kapitel 10 mit einer Zusammenfassung der zentralen Ergebnisse und einem Ausblick auf zukünftige Forschungsthemen, die aus den Erkenntnissen dieser Arbeit abgeleitet werden können.

2 Grundlagen und Abgrenzung der Untersuchung

Um die Ausrichtung dieser Arbeit zu verdeutlichen und das Verständnis der Autorin transparent zu machen, ist es wichtig, zunächst grundlegende Begriffe zu definieren und einzuordnen, die für diese Arbeit relevant sind. Im ersten Schritt werden die für diese Arbeit relevanten Begriffe im Zusammenhang mit Low-Code erläutert und definiert (siehe Kapitel 2.1). Darauffolgend wird das technische Verständnis von Informationssystemen und Architekturen vertieft und im Kontext dieser Arbeit werden spezifische Begriffe festgelegt (siehe Kapitel 2.2). Abschließend wird das Management von IT-Architekturen beleuchtet, wobei insbesondere organisatorische Aspekte berücksichtigt werden (siehe Kapitel 2.3). Schließlich wird der Umfang der Untersuchung durch die definierten Begriffe und Themen eingegrenzt und konkretisiert (siehe Kapitel 2.4).

2.1 Low-Code

Im nachfolgenden Abschnitt wird ein Begriffsverständnis für Low-Code geschaffen, indem zuerst auf die Unterscheidung zwischen No-Code und Low-Code eingegangen wird (siehe Kapitel 2.1.1). Im Anschluss werden Low-Code-Plattformen spezifiziert (siehe Kapitel 2.1.2) und es wird auf den Low-Code-Entwicklungsprozess eingegangen (siehe Kapitel 2.1.3). Zum Abschluss wird zwischen den Kernanwender*innen der Low-Code-Plattformen, den Citizen Developern und Pro Developern differenziert und diese werden für die vorliegende Arbeit präzise definiert (siehe Kapitel 2.1.4).

2.1.1 Low-Code und No-Code

Ursprünglich lässt sich Low-Code von der Programmiersprache der vierten Generation und dem Konzept des „Rapid Application Development" (RAD) ableiten, die ein höheres Abstraktionsniveau im Vergleich zu früheren Programmiersprachen anstrebten (s. WASZKOWSKI 2019, S. 376). Low-Code gehört zu den deklarativen Programmiersprachen, die das Gegenstück zur imperativen Programmierung darstellen. Bei deklarativen Sprachen wird lediglich die Spezifikation von Objekten beschrieben und die Entwickelnden beschreiben den Endzustand ohne genau zu spezifizieren, wie dieses Ziel erreicht werden soll (s. KHORRAM ET AL. 2020, S. 1; STYCZYNSKI ET AL. 2017, S. 67-68).

Das Hauptziel der Low-Code-Entwicklung besteht darin, mit einer reduzierten Komplexität und Menge an Programmiercode im Vergleich zur herkömmlichen Programmierung zu arbeiten. Gleichzeitig ermöglicht Low-Code umfangreichere und spezifischere Lösungen im Vergleich zur No-Code-Methodik. Low-Code wird oft mit dem Ansatz des Model-Driven Software Development (MDSD) verglichen, da beide Konzepte darauf abzielen, die Entwicklungsgeschwindigkeit zu erhöhen (s. STAHL U. VÖLTER 2006, S. 13). Es gibt jedoch Unterschiede zwischen den Methoden, da bei Low-Code die Entwickelnden oft keine Kenntnis darüber haben, wie der Code durch die Entwicklungssoftware im Hintergrund erzeugt wird und welche Programmier-Frameworks verwendet werden. Außerdem ist es bei der Low-Code-Entwicklung in der Regel nicht

möglich, die Programmiersprache zu ändern, mit der die Anwendung programmiert wird. Es kommt auch vor, dass bestimmte Teile des Programmcodes nachträglich angepasst oder manuell von Entwickler*innen vervollständigt werden müssen (s. DA CRUZ ET AL. 2021, S. 2; CABOT 2020, S. 2). Tabelle 1 zeigt die Einordnung von Low-Code in die Generationen von Programmiersprachen und den Bezug zur modellgetriebenen Softwareentwicklung.

Tabelle 1: Klassifizierung der fünf Generationen von Programmiersprachen (s. WASZKOWSKI 2019)

Generationen von Programmiersprachen	Merkmale
1GL Maschinensprache	Binär, komplexere Programmierung kaum realisierbar
2GL Assemblersprache	Umwandlung in ausführbare Maschinensprache
3GL Prozessuale Programmierung	Strukturierte und nutzerfreundliche Programmierung (z. B. Java, C++)
4GL Deklarative Programmierung	Modellgetriebene Softwareentwicklung (Low-Code-Programmierung)
5GL Künstliche Intelligenz	Automatisierte Weiterentwicklung einer Software

Der Ansatz der No-Code-Softwareentwicklung ermöglicht die Erstellung von Anwendungen, ohne dass Programmierfähigkeiten erforderlich sind, da die Notwendigkeit des Schreibens von Code vollständig entfällt (s. DA CRUZ ET AL. 2021, S. 2). In diesem Konzept ermächtigt allein die Fähigkeit der Entwickelnden, logisch und abstrakt zu denken, zur Konstruktion von Softwarelösungen. Es fällt jedoch auf, dass auch erfahrene Softwareentwickler*innen gelegentlich auf diesen Ansatz zurückgreifen, um Zeit zu sparen. Dies ist besonders relevant, da professionelle Entwickler oft auf eine über Jahre erlernte Programmiersprache beschränkt sind, was einen Wechsel zu einer anderen Softwaretechnologie mit erheblichem Aufwand verbindet. No-Code bietet hier die Möglichkeit, den Aufwand für das Erlernen einer neuen Programmiersprache zu minimieren (vgl. MOSKAL 2021, S. 54-55), und ermöglicht beispielsweise Back-End-Entwicklern den Zugang zur Front-End-Entwicklung.

Es ist jedoch zu beachten, dass der durch die No-Code-Entwicklungsplattform im Hintergrund generierte Programmcode selten anpassbar oder erweiterbar ist. Daher weisen die entwickelten Anwendungen in der Regel nur eine geringe Integrationsfähigkeit und Skalierbarkeit auf, insbesondere im Kontext unternehmensweiter Softwareanwendungen (s. JEDNASZEWSKI 2024). Im Rahmen dieser Arbeit wird der Begriff No-Code Softwareentwicklung nicht weiterverwendet und stattdessen der Begriff der Low-Code-Softwareentwicklung für beide Softwareentwicklungskonzepte genutzt. Je nach Anwendungsfalltyp kommt es bei der Low-Code-Entwicklung zu mehr oder weniger Nutzung des herkömmlichen Programmiercode (siehe Kapitel 5.3).

2.1.2 Low-Code-Plattformen

Low-Code-Plattformen ermöglichen es den Mitarbeiter*innen eines Unternehmens, auf die Funktionalitäten der Low-Code-Programmierung zuzugreifen. Diese Plattformen abstrahieren den Entwicklungsprozess mithilfe visueller Werkzeuge und führen im Hintergrund die Transformation des Modells in ausführbaren Quellcode durch. Mit Low-Code-Plattformen können Anwendungen erstellt werden, die verschiedene Prozesse im Unternehmen digitalisieren und optimieren. Die entwickelten Anwendungen können auf unterschiedlichen Endgeräten und in Webbrowsern verwendet werden (s. DEGENHARDT 2020). Die sogenannte visuelle Modellierung ermöglicht es Nutzern, über eine Drag-&-Drop-Funktion Funktionsbausteine in Bildschirmansichten einzufügen und diese zu verbinden. Abbildung 2-1 zeigt als Beispiel die Low-Code-Entwicklungsoberfläche der Open-Source-Lösung „Node-RED“. Die verschiedenen Funktionsbausteine auf der linken Seite können links ausgewählt und in der mittleren Ansicht bearbeitet und miteinander verbunden werden. Auf diese Weise können Anwendungen erstellt werden. Die verwendeten Daten werden häufig in Clouds abgespeichert, um eine ortsunabhängige Verwendung der Anwendungen zu ermöglichen (s. ADRIAN ET AL. 2020b).

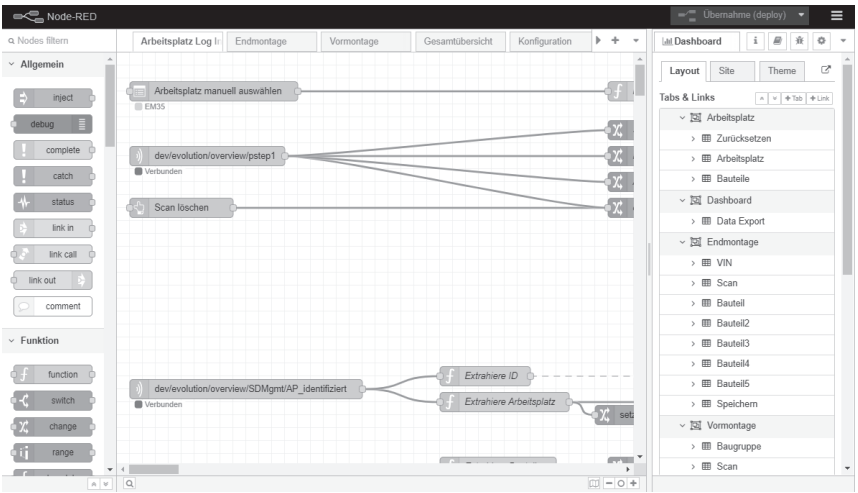

Abbildung 2-1: Low-Code-Entwicklungsoberfläche am Beispiel der Open-Source-Lösung "Node-RED" (eigene Darstellung)

Die verschiedenen Plattformen auf dem Markt unterscheiden sich in ihren Funktionalitäten. In der Softwareentwicklung kann der Begriff „Funktion" aufgrund der Übersetzung aus dem Englischen ins Deutsche verschiedene Bedeutungen haben. Daher soll dieser Begriff in diesem Abschnitt konkret abgegrenzt werden. Die Funktion (im Englischen: Feature) einer Softwareanwendung wurde vom Verband Institute of Electrical and Electronics Engineers (IEEE) als „a distinguishing characteristic of a software item (e.g., performance, portability, or functionality)“ definiert (vgl. METZGER U. POHL

2014) und bildet das Fundament des „Feature-Oriented Software Developments“. Der Grundgedanke besteht darin, ein Softwaresystem in Bezug auf die von ihm bereitgestellten Funktionen zu zerlegen (s. APEL U. KÄSTNER 2009). Dieser Definition soll in der vorliegenden Arbeit nicht gefolgt werden. Stattdessen soll sich die Funktion (im Englischen: Function) an dem Konzept der Routinen orientieren, das eine Grundtechnik im Programmieren darstellt. Nach diesem von KNUTH beschriebenen Ansatz werden bestimmte Code-Blöcke eines Programms nicht kontinuierlich von Grund auf entwickelt, sondern im Entwicklungsprozess immer wieder an anderen Stellen für die Ausführung einer ähnlichen Aufgabe eingesetzt (s. KNUTH 1997, S. 186-201). Eine Funktion ist damit ein wiederverwendbares Entwicklungselement, beispielsweise in Form von User-Interface (UI) Elementen. Somit wird die Funktionalität beschrieben, die bei den jeweiligen Entwicklungsanwendungen den Entwickelnden durch die Funktionsbausteine bereitgestellt wird. Abbildung 2-2 gibt einen Überblick über den Einsatz von Low-Code-Plattformen in Unternehmen.

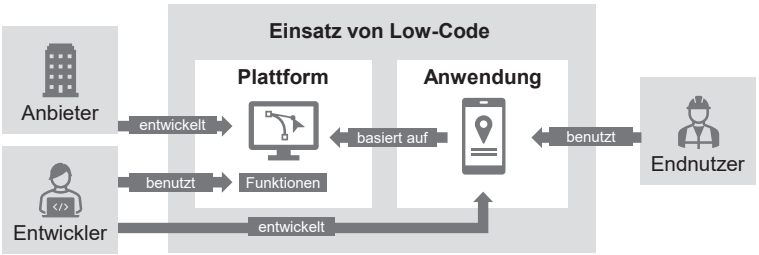

Abbildung 2-2: Einsatz von Low-Code in Unternehmen (eigene Darstellung)

Abhängig von dem zuvor beschriebenen Funktionsbereich werden die Low-Code-Plattformen für verschiedene Anwendungsbereiche bzw. Branchen eingesetzt. Einige Plattformen richten sich an Nutzer*innen, die ohne umfangreiche Programmierfähigkeiten Geschäftsanwendungen erstellen können, während andere Plattformen speziell für erfahrene Softwareentwickler*innen geeignet sind und darauf abzielen, die Dauer und Kosten des Entwicklungsprozesses zu reduzieren (s. RYMER U. RICHARDSON 2016, S. 4). Im Folgenden soll ein Grundverständnis für den Entwicklungsprozess mit Low-Code-Plattformen geschaffen werden.

2.1.3 Low-Code-Entwicklungsprozess

Softwareentwicklung, ist ein umfassender Prozess, der die Planung, Definition, Gestaltung und Umsetzung eines Produkts umfasst, das sowohl die geforderten Qualitätsmerkmale erfüllt als auch die Kundenwünsche berücksichtigt (s. BALZERT 2009, S. 39). Dieser Prozess bildet die Grundlage für den Lebenszyklus der Softwareentwicklung (SDLC, Software Development Life Cycle). Es handelt sich um einen strukturierten Ablauf, der es Entwicklerteams ermöglicht, effizient zusammenzuarbeiten, um ein Softwareprodukt innerhalb eines festgelegten Zeitrahmens und Budgets mit allen erforderlichen Funktionen zu erstellen (s. ACHARYA U. SAHU 2020). SDLC-Modelle

verwenden einen schrittweisen Ansatz, um den Softwareentwicklungsprozess zu durchlaufen (s. GURUNG ET AL. 2020). Es gibt verschiedene SDLC-Modelle mit jeweiligen Vor- und Nachteilen (s. ALAZZAWI ET AL. 2023), die je nach den individuellen Anforderungen ausgewählt werden (s. GURUNG ET AL. 2020). Es wird das SDLC-Modell ausgewählt, das am besten zum jeweiligen Softwareprojekt passt (s. KUMAR U. RASHID 2018). Tabelle 2 zeigt eine Übersicht verschiedener SDLC-Modelle. Grundsätzlich lässt sich zwischen dem traditionellen und dem agilen Ansatz in der Softwareentwicklung unterscheiden (s. ALAZZAWI ET AL. 2023).

Tabelle 2: Übersicht verschiedener SDLC Modelle (eigene Darstellung i. A. a. ALAZZAWI ET AL. 2023)

Traditioneller SDLC-Ansatz	**Agiler SDLC-Ansatz**
Wasserfall-Modell	eXtreme Programming
Interaktives Modell	Scrum-Modell
Spiralmodell	Feature Driven Development
V-Modell	Kanban-Modell
Big Bang-Modell	

Die agilen Ansätze, bekannt für den Fokus auf Flexibilität, schnelle Entwicklung und iteratives Feedback, stehen in enger Übereinstimmung mit den Prinzipien der Low-Code-Entwicklung (s. AL ALAMIN ET AL. 2020). Der Low-Code-Ansatz erleichtert den schnellen Aufbau von Anwendungen, ermöglicht eine einfache Sammlung von Nutzungsfeedback und unterstützt die zügige Umsetzung von Änderungen. Diese Arbeitsweise ist von Natur aus agil, da sie kontinuierliche Lieferung, schrittweise Verbesserungen und gesteigerte Kundenzufriedenheit fördert (s. ROKIS U. KIRIKOVA 2022). Der Vergleich zwischen der Low-Code-Entwicklung und des agilen Modells ist in Abbildung 2-3 dargestellt (s. AL ALAMIN ET AL. 2020). Der äußere Kreis beschreibt die Entwicklungsphasen des agilen Modells, während der innere Kreis die typischen Entwicklungsphasen der Low-Code-Entwicklung darstellt.

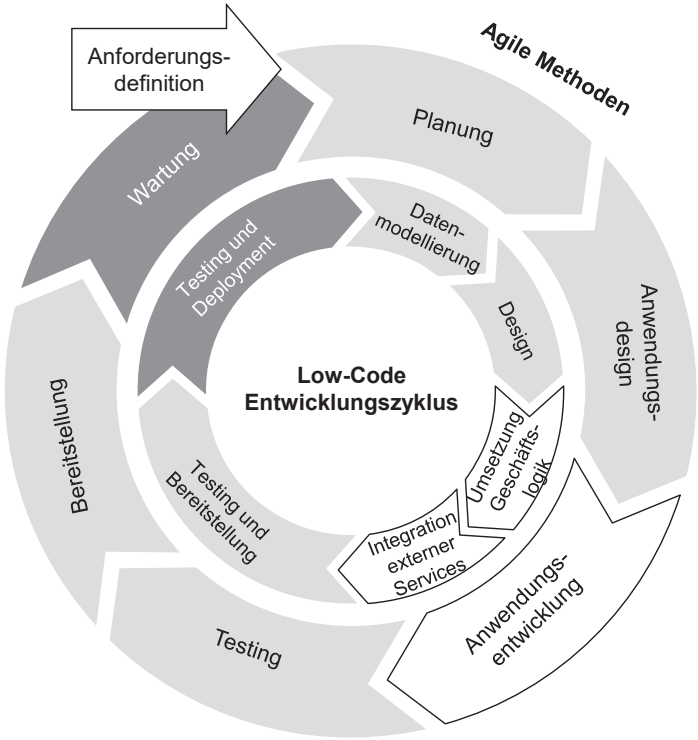

Abbildung 2-3: Vergleich agiles Modell und Low-Code-Entwicklungszyklus (eigene Darstellung i. A. a. Al Alamin et al. 2020)

Das Hauptziel von Low-Code ist es, Anwendungen zu erstellen, Nutzungsfeedback einzuholen und Änderungen schnell zu integrieren. Daher ergänzen sich die agile Entwicklungsmethodik und die Low-Code-Softwareentwicklung gut, da beide auf Kundenzufriedenheit und kontinuierliche, inkrementelle Lieferung abzielen (s. Al Alamin et al. 2020). Low-Code-Plattformen übernehmen viele Herausforderungen der Anwendungsentwicklung, wodurch einige agile Entwicklungsphasen bei Low-Code kürzere Zeitspannen im Vergleich zur traditionellen Softwareentwicklung haben (s. Rokis u. Kirikova 2022).

In den folgenden Kapiteln werden die Phasen der Softwareentwicklung im Detail vorgestellt und ihre Bedeutung für die Softwareentwicklung erörtert.

Anforderungsdefinition

Die Anforderungsdefinition ist der erste Schritt, um die Erwartungen der Benutzer*innen an die zu entwickelnde Software zu identifizieren (s. Rokis u. Kirikova 2022). Das Hauptziel dieser Phase ist es, die Anforderungen und Erwartungen der Benutzer*innen zu verstehen (s. Leloudas 2023). Die Low-Code-Entwickelnden weisen

unterschiedliche technische und fachliche Kenntnisse auf (siehe Kapitel 2.1.4). So kann eine Anforderung, wie sie in Abbildung 2-4 zu sehen ist, aus zwei Perspektiven beschrieben werden – aus der fachlichen Perspektive, dem Problemraum oder aus der technischen Perspektive, dem Lösungsraum (s. HERRMANN 2022, S. 3-4).

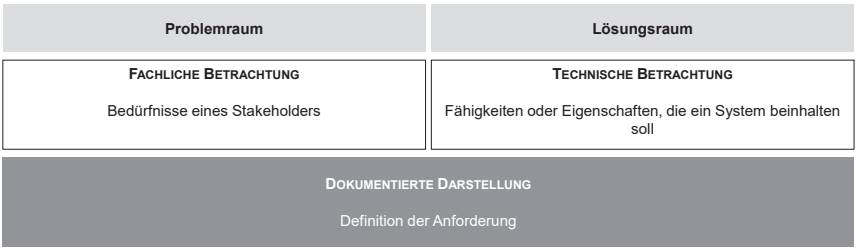

Abbildung 2-4: Problemraum und Lösungsraum in der Anforderungsdefinition (eigene Darstellung i. A. a. HERRMANN 2022, S. 3-4)

Neben der fachlichen und technischen Perspektive können Anforderungen funktional und nicht-funktional sein.

- **Funktionale Anforderungen**: Funktionale Anforderungen beschreiben die spezifischen Funktionen oder Verhaltensweisen, die ein System ausführen soll. Sie definieren, was das System tun soll, einschließlich Benutzerinteraktionen, Datenverarbeitung und -speicherung, Berechnungen und anderen spezifischen Funktionalitäten. Funktionale Anforderungen sind oft klar definiert und überprüfbar. Sie bilden die Grundlage für die Entwicklungsziele des Systems (s. RAHIMI ET AL. 2020).
- **Nicht-funktionale Anforderungen**: Im Gegensatz dazu beziehen sich nicht-funktionale Anforderungen auf die Qualität und Eigenschaften des Systems als Ganzes. Sie umfassen Aspekte wie Leistung, Sicherheit, Benutzerfreundlichkeit, Zuverlässigkeit und Wartbarkeit. Nicht-funktionale Anforderungen sind oft schwieriger zu quantifizieren und zu überprüfen als funktionale Anforderungen. Sie haben einen wesentlichen Einfluss auf die Benutzererfahrung und die Gesamtwahrnehmung der Softwarequalität (s. UMAR U. KHAN 2011).

Nicht-funktionale Anforderungen sollten frühzeitig in den Entwicklungsprozess integriert werden, um sicherzustellen, dass das Endprodukt den Benutzererwartungen entspricht. Die Berücksichtigung von nicht-funktionalen Anforderungen sollte systematisch und im Einklang mit den funktionalen Anforderungen erfolgen, um eine ausgewogene und effektive Softwarelösung zu gewährleisten (s. RAHARJA U. SIAHAAN 2019). Nicht-funktionale Anforderungen wie Architekturvorgaben können mithilfe automatisierter Tests und Check-Tools kommuniziert und verifiziert werden (s. LAGERSTEDT 2014). In dieser Arbeit werden sowohl die fachliche als auch die technische Betrachtung der Anforderungsdefinition berücksichtigt, zudem wird auch zwischen funktionalen und nicht-Funktionalen Anforderungen unterschieden (siehe Kapitel 6.2.4). Die gesammelten Anforderungen dienen als Basis für das Design und die Datenmodellierung.

Design und Datenmodellierung

In dieser Phase wird im Detail festgelegt, wie ein System die bei der Systemanalyse ermittelten Anforderungen erfüllen soll und das System wird logisch entworfen (s. MUCHHAL ET AL. 2023). Beim Design und der Datenmodellierung kann zwischen zwei Hauptansätzen unterschieden werden (s. SAHAY ET AL. 2020):

- **Benutzeroberfläche zu Datenmodellierung**: Bei diesem Ansatz beginnen die Entwickelnden mit dem Erstellen der Benutzeroberfläche und verknüpft sie dann mit den benötigten Datenquellen. Formulare und Seiten werden zuerst definiert, gefolgt von der Spezifikation von Geschäftslogikregeln und Workflows. Dies führt schließlich zur Integration externer Dienste vor der Bereitstellung der Anwendung.
- **Datenmodellierung zu Benutzeroberfläche**: Hierbei handelt es sich um einen datengetriebenen Ansatz, der mit der Datenmodellierung beginnt und anschließend die Benutzeroberfläche der Anwendung erstellt. Danach folgt die Spezifikation von Geschäftslogikregeln und Workflows und schließlich, falls notwendig, die Integration externer Dienste vor der Bereitstellung der Anwendung.

In beiden Ansätzen können die Spezifikation von Geschäftslogikregeln, Workflows und die Integration externer Dienste je nach Stil der Entwickelnden angepasst und vertauscht werden (s. SAHAY ET AL. 2020). Es gibt viele Möglichkeiten, ein Dokument mit detaillierten Entwurfsspezifikationen zu präsentieren (s. MUCHHAL ET AL. 2023). Das Ergebnis der Designphase ist eine detaillierte Designspezifikation, die als Grundlage für die Entwicklung dient (s. LELOUDAS 2023).

Entwicklung

In der Entwicklungsphase wird die Software auf der Grundlage der in den vorangegangenen Phasen gesammelten Anforderungen und Designspezifikationen entworfen und entwickelt (s. LELOUDAS 2023). In dieser Phase wird die eigentliche Software entwickelt. Dazu zählen das Anpassen der Benutzeroberfläche, das Implementieren der Geschäftslogik, das Integrieren von Diensten Dritter oder das Lösen komplexerer Entwicklungsprobleme, die Zugriff auf den Code erfordern (s. ROKIS U. KIRIKOVA 2022). Die Verwaltung von Workflows zwischen verschiedenen Formularen oder Seiten wird durch visuell basierte Workflows umgesetzt (s. ROKIS U. KIRIKOVA 2022).

Während der Entwicklungsphase ist es wichtig, dass Kodierungsstandards und bewährte Verfahren eingehalten werden, um sicherzustellen, dass der Softwarecode wartbar, wiederverwendbar und in Zukunft leicht erweiterbar ist (s. LELOUDAS 2023). Außerdem wird sichergestellt, dass der Code gut dokumentiert ist und verständlich ist, warum bestimmte Abläufe auf eine bestimmte Weise kodiert werden (s. MUCHHAL ET AL. 2023). Darüber hinaus werden Tests an der entwickelten Anwendung durchgeführt, um zu überprüfen, ob alle festgelegten Anforderungen erfüllt sind (s. KHORRAM ET AL. 2020).

Da der Fokus dieser Arbeit nicht auf der operativen Entwicklung von Anwendungen liegt, sondern vielmehr darauf, wie die entwickelten Anwendungen nachhaltig in der

Geschäftsarchitektur verankert werden können, wird der operative Entwicklungsprozess soweit vereinfacht dargestellt, dass die Design-, die Datenmodellierung-, die Entwicklungs- und Testing-Phase unter dem Begriff „Lösungsentwicklung“ zusammengefasst werden.

Bereitstellung

Die Entwicklung und Pflege von umfassenden Tests (einschließlich Unit-Tests, Integrationstests und Akzeptanztests), um die Funktionalität und Qualität der Software kontinuierlich zu überprüfen, verhindert den Aufbau von technischen Schulden in der Softwareentwicklung (s. FREIRE ET AL. 2020). Technische Schulden sind metaphorisch für unreife, unvollständige oder unzureichende Artefakte im Software-Entwicklungszyklus, die langfristig höhere Kosten und niedrigere Qualität verursachen. Diese Artefakte beeinflussen nachfolgende Entwicklungs- und Wartungsaktivitäten und können als eine Art Schuld angesehen werden, die die Systementwickler*innen dem System schulden (s. SEAMAN U. GUO 2011).

Die Bereitstellungsphase ist die Phase, in der die entwickelte Software den Nutzer*innen übergeben wird. Sie umfasst die Installation der Software in der Technologie-Architektur, ihre Konfiguration und die Sicherstellung, dass sie wie erwartet funktioniert (s. LELOUDAS 2023). Typischerweise integrieren Low-Code-Plattformen verschiedene Komponenten und Hilfsprogramme für die Bereitstellung, um den Einsatz leicht und erfolgreich durchzuführen (s. AL ALAMIN ET AL. 2020).

Eine besondere Rolle bei der Bereitstellung von Low-Code-Plattformen spielt das Konfigurationsmanagement, da sich die Low-Code-Plattformen ständig weiterentwickeln (s. ROKIS U. KIRIKOVA 2022). Die Weiterentwicklung führt zu einer Serie von Systemversionen, die jeweils gewartet und verwaltet werden müssen. Konfigurationsmanagement spielt eine entscheidende Rolle, indem es Richtlinien, Prozesse und Werkzeuge zur Verwaltung dieser Veränderungen bietet. Es hilft, den Überblick über Änderungen und Versionen zu behalten, um Fehlanpassungen und Verluste von Quellcode zu vermeiden. Das Konfigurationsmanagement umfasst vier eng miteinander verbundene Aktivitäten (siehe Abbildung 2-5, SOMMERVILLE 2016, S. 731-732)

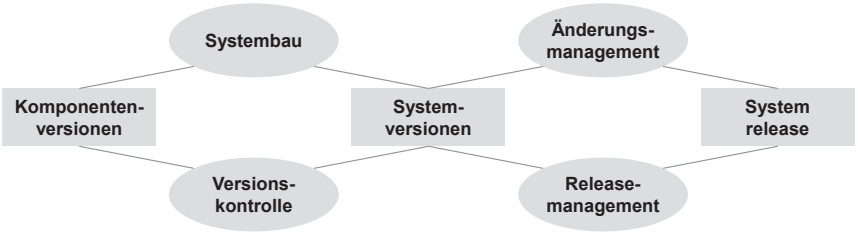

Abbildung 2-5: Aktivitäten des Konfigurationsmanagements (eigene Darstellung i. A. a. SOMMERVILLE 2016, S. 731-732)

- **Systembau**: Der Prozess des Zusammenstellens von Programmkomponenten, Daten und Bibliotheken. Diese müssen kompiliert und verlinkt werden, um ein ausführbares System zu erstellen.
- **Versionskontrolle**: Beinhaltet die Überwachung der verschiedenen Versionen von Systemkomponenten und stellt sicher, dass Änderungen, die von verschiedenen Entwicklern an Komponenten vorgenommen werden, sich nicht gegenseitig negativ beeinflussen.
- **Änderungsmanagement**: Beinhaltet die Überwachung von Anfragen für Änderungen an ausgelieferter Software von Kund*innen und Entwickler*innen sowie das Ausarbeiten der Kosten und Auswirkungen dieser Änderungen. Zudem werden die Entscheidungen beschrieben, ob und wann die Änderungen umgesetzt werden sollten.
- **Release-Management**: Beinhaltet die Vorbereitung der Software für die externe Veröffentlichung und die Überwachung der Systemversionen, die für den Kundengebrauch freigegeben wurden.

Nach der Bereitstellung der Software erfolgt die Wartung.

Wartung

Die Wartungsphase ist entscheidend für die Bereitstellung und Pflege von Softwareprodukten (s. LELOUDAS 2023). Sie umfasst verschiedene Aktivitäten und Aufgaben, die notwendig sind, um ein bestehendes Softwareprodukt zu ändern und die Integrität zu wahren (s. ISO 14764). Nach der Bereitstellung der Anwendung ist Wartung notwendig, um eine kontinuierliche Verfügbarkeit der Anwendungen zu gewährleisten, indem Änderungen implementiert werden (s. ROKIS U. KIRIKOVA 2022).

Wie in Abbildung 2-6 dargestellt, können die Änderungen während dieser Phase als Korrektur oder Erweiterung klassifiziert werden und fallen in die Kategorien der korrigierende, präventive, adaptive oder perfektionierende Wartung (s. ISO 14764).

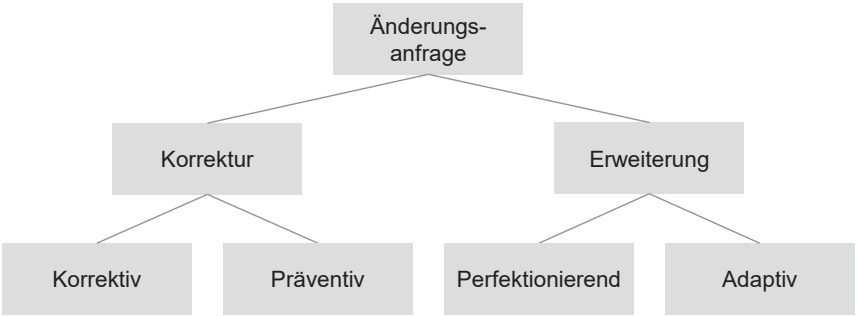

Abbildung 2-6: Änderungsanfragen bei der Bereitstellung und Pflege von Softwareprodukten (eigene Darstellung i. A. a. ISO 14764)

Die Korrektur ist die reaktive Änderung einer Anwendung, die nach der Auslieferung vorgenommen wird, um festgestellte Probleme zu beheben. Die Korrektur repariert die

Anwendung, um die Anforderungen zu erfüllen. Die Erweiterung hingegen ist eine Änderung eines bestehenden Softwareprodukts, um eine neue Anforderung zu erfüllen. Eine Wartungserweiterung ist keine Softwarekorrektur (s. ISO 14764). Die vier Wartungskategorien können wie folgt beschrieben werden:

- **Korrektiv**: Bei der korrektiven Wartung handelt es sich um die Behebung von Softwarefehlern. Softwarefehler können unterschiedlicher Art sein. Es handelt sich um Kodierungsfehler, Designfehler und Fehler bei den Anforderungen. Kodierungsfehler werden von Programmierern korrigiert. Die Fehler bei den Anforderungen sind sogar noch wichtiger, denn die Software wird gebaut, um bestimmte Anforderungen zu erfüllen, und das System muss neu erstellt werden, wenn diese Anforderungen falsch sind (s. UMUDOVA 2019). Ein Teil der korrektiven Wartung ist die Notwartung. Die Notwartung ist eine außerplanmäßige Änderung, die durchgeführt wird, um ein System vorübergehend betriebsbereit zu halten, bis eine korrektive Wartung erfolgt (s. ISO 14764).
- **Präventiv**: Diese Wartung zielt darauf ab, die Dauerhaftigkeit oder Zuverlässigkeit der Software zu erhöhen, um zukünftige Probleme zu verhindern (s. UMUDOVA 2019).
- **Perfektionierend**: Die perfektionierende Wartung beschäftigt sich mit dem Hinzufügen oder Ändern von Funktionen zum System. In dieser Kategorie werden Änderungen vorgenommen, um ein effizienteres und funktionelleres System in Übereinstimmung mit den sich ändernden Anforderungen zu schaffen (s. UMUDOVA 2019). Perfektionierende Wartung bietet Erweiterungen für Benutzer*innen, Verbesserung der Dokumentation und Umkodierung zur Verbesserung der Softwareleistung, Wartbarkeit oder anderer Softwareeigenschaften (s. ISO 14764).
- **Adaptiv**: Bei der adaptiven Wartung wird die Software an unterschiedliche Betriebsumgebungen angepasst. Sie ist einer der wichtigsten Faktoren, damit sich die Software während ihres Lebenszyklus an verschiedene Umgebungen anpasst (s. UMUDOVA 2019).

In dieser Arbeit wird zwischen dem geplanten Betrieb und dem reaktiven Support für Serviceanfragen und Störungen unterschieden. Während zu dem geplanten Betrieb der gesamte Bereitstellungsprozess sowie die adaptive und perfektionierende Erweiterung der Low-Code-Anwendung zählen, zählen die korrektive und präventive Wartung zum Support der Low-Code-Anwendung. Im Support kann das Softwareentwicklungsteam auch technische Unterstützung für Endbenutzer*innen bieten und Schulungen durchführen, um sicherzustellen, dass die Benutzer*innen die Software optimal nutzen können (s. LELOUDAS 2023). Abbildung 2-7 zeigt wie die Entwicklungsphasen in dieser Arbeit betrachtet werden.

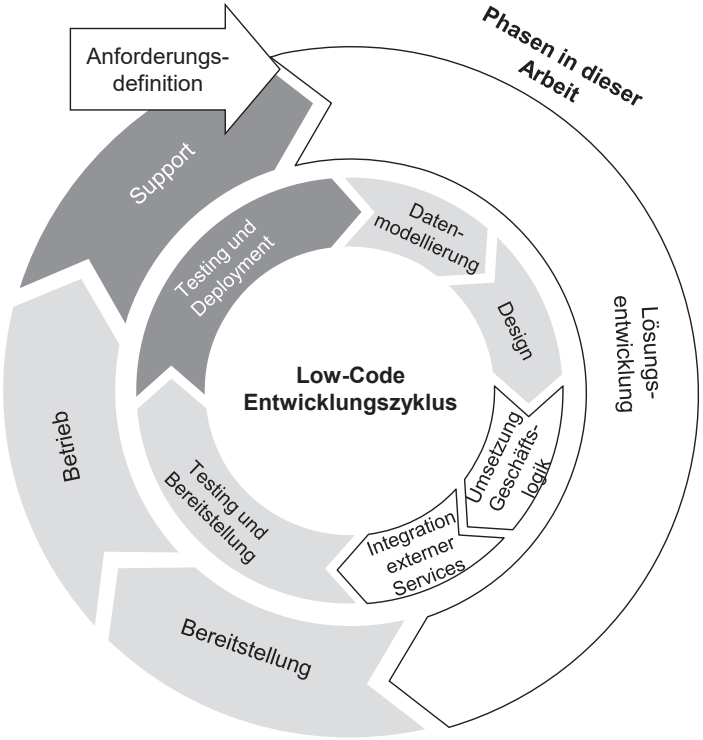

Abbildung 2-7: Betrachtete Entwicklungsphasen in dieser Arbeit (eigene Darstellung i. A. a. Al Alamin et al. 2020)

Zusammenfassend lässt sich festhalten, dass der Low-Code-Entwicklungsprozess den Prozess der Softwareentwicklung vereinfacht, während er gleichzeitig die Einhaltung der üblichen Phasen der Softwareentwicklung gewährleistet. Dieser Ansatz ermöglicht es auch nicht-technischen Nutzern*innen, den sogenannten „Citizen Developers", Anwendungen und Lösungen zu entwickeln, ohne über umfangreiche Programmierfähigkeiten verfügen zu müssen (vgl. Lebens et al. 2022; Ullrich, C., Lata, T., Geyer-Klingeberg, J 2020, S. 1). Diese Form der Entwicklung wird als „Citizen Development“ bezeichnet.

2.1.4 Citizen Development

Gartner spezifiziert den Begriff des „Citizen Development“ weiter und beschreibt, dass ein „Citizen Developer" bei der Entwicklung Softwaretools verwendet, die nicht aktiv von der IT oder der Geschäftseinheit verboten sind. Außerdem sind die „Citizen Developer" Mitarbeitende, die einer anderen Geschäftseinheit als der IT-Abteilung unterstellt sind. In der Literatur ist für die Bezeichnung der Mitarbeitenden als „Citizen

Developer" bisher keine fachliche Qualifikation oder zeitliche Zuordnung vorgeschrieben, jedoch müssen „Citizen Developer" offizielle Angestellte eines Unternehmens sein (vgl. GARTNER 2022). Das Phänomen der „Citizen Developer" soll die Softwareanfragen an technische Softwareentwickler*innen entweder überflüssig machen oder wenigstens vereinfachen. Es ermutigt IT-affine Angestellte, ihren Bedarf an Softwarelösungen ohne oder mit weniger Unterstützung durch erfahrene Programmierer*innen zu erfüllen (s. NG'AMBI 2020).

GARTNER unterscheidet bei „Citizen Developers" zwischen zwei Typen: den „Power Usern" und den „End Usern". Zwar sind beide Entwicklertypen keine Vollzeit-Programmierer*innen, jedoch unterscheiden sie sich erheblich durch ihre bevorzugte Entwicklungsmethodik. „End User" erstellen über Low-Code-Programmierung Anwendungen, die insbesondere ihre persönliche Effizienz am Arbeitsplatz steigern und ihren individuellen Arbeitsalltag erleichtern. „Power User" sind hingegen software-affinere Mitarbeitende, die teilweise bereits geringfügige Programmiererfahrungen mitbringen. Die Applikationsentwicklung erfolgt bei diesem Entwicklertyp in der Regel über Low-Code-Plattformen, wodurch komplexere Softwareanwendungen erstellt werden, die für ein gesamtes Team oder eine Arbeitsgruppe relevant sind (s. WONG 2019). Beiden Typen von „Citizen Developers" wird jedoch eine Domänen-Expertise unterstellt. Diese beschreibt ein tiefes Wissen der Mitarbeitenden in ihrem spezifischen Fachgebiet, einschließlich eines logischen Verständnisses der Geschäftsprozesse in der jeweiligen Abteilung (s. COSTABILE ET AL. 2003, S. 1).

Auch in dieser Arbeit soll die Kompetenz im Hinblick auf die IT-Kenntnisse und die Programmiererfahrung in die beschriebenen „Citizen Developer"-Typen unterteilt werden. Der Verständlichkeit halber wird jedoch der „End User" als Mitarbeitender mit keinen Programmierfähigkeiten und der „Power User" als Mitarbeitender mit wenig Programmierfähigkeiten beschrieben.

Erfahrene Entwickler*innen (Professional Developer oder kurz Pro Developer) sind Mitarbeitende eines Unternehmens mit einer professionellen Ausbildung oder Berufserfahrung im Bereich der Softwareentwicklung. Im traditionellen Verständnis verwenden diese Pro Developer textbasierte Programmiersprachen wie beispielsweise C++, Java oder Python (s. HIRZEL 2022). GARTNER klassifiziert Pro Developer grundsätzlich als Vollzeitentwickler*innen und vertieft diese Klassifizierung ergänzend mit zwei Kategorien. Die Business-Led Pro Developer entwickelt abteilungsweite Anwendungen auf Basis von professionellem Code, nutzen jedoch auch Low-Code, um Zeit zu sparen. Unternehmensweite Softwareanwendungen, die ausschließlich auf professionellem Code basieren, werden dagegen von Enterprise IT Pro Developern erstellt (s. WONG 2019). Diese Unterscheidung ist für die vorliegende Arbeit jedoch nicht relevant. Erfahrene Entwickler*innen werden hier als Softwareentwickler*innen einer bestimmten Fachabteilung oder der IT-Abteilung definiert, die aus Effizienzgründen auf Low-Code zurückgreifen können, sich jedoch auch für die textbasierte Entwicklung eignen. Abbildung 2-8 zeigt die für diese Arbeit definierten Rollen im Citizen Development.

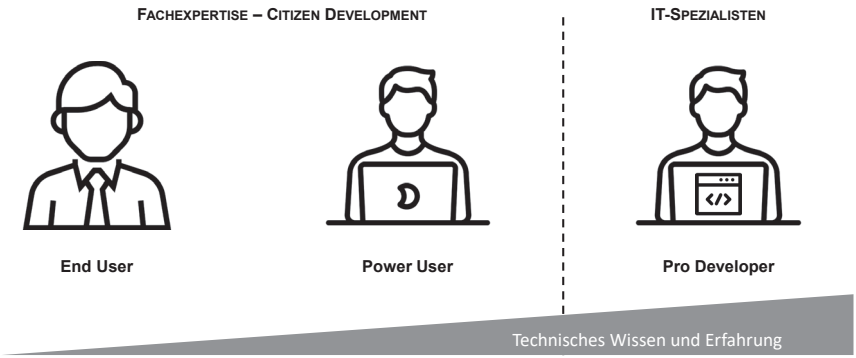

Abbildung 2-8: Verschiedene Rollen im Citizen Development (eigene Darstellung i. A. a. GARTNER 2022)

Nachdem für diese Arbeit ein Grundverständnis für Low-Code und den Low-Code-Entwicklungsprozess geschaffen wurde, sollen nun die gängigen technischen Begriffe in Bezug auf die Informationssysteme und Architekturen in Organisationen erläutert werden.

2.2 Informationssysteme und Architekturen

In dieser Arbeit wird die Implementierung und Betrieb von Low-Code-Anwendung in Unternehmen untersucht. Diese Anwendungen werden mithilfe von Daten entwickelt und anschließend in die bestehende IT-Architektur des Unternehmens integriert. In diesem Zusammenhang ist es wesentlich, die technischen Konzepte und Begriffe, die bei der Implementierung und dem Betrieb der Anwendungen eine Rolle spielen, detailliert zu erläutern. Dazu wird zunächst der Begriff der Informationen und Daten beschrieben (siehe Kapitel 2.2.1), die von Informationssystemen bzw. Anwendungen verwaltet werden (siehe Kapitel 2.2.2). Diese Anwendungen stehen in Verbindung mit anderen Systemen über Schnittstellen (siehe Kapitel 2.2.3). Die Gesamtheit der so strukturierten Informationssysteme in Unternehmen bildet die IT-Architektur (siehe Kapitel 2.2.4).

2.2.1 Daten und Informationen

Im allgemeinen Sprachgebrauch erfolgt oft keine präzise Unterscheidung zwischen den Begriffen „Daten" und „Informationen" (s. KRCMAR 2015, S. 11-13; HOFFMANN 2018, S. 13; LEHNER ET AL. 2008, S. 201). Der Begriff „Daten" wird verwendet, wenn Zeichen in einen regelbasierten, formalisierten Zusammenhang gebracht werden. Sobald Daten mit zusätzlichem Kontext angereichert werden, erhalten sie Bedeutung, wodurch Informationen entstehen. Wenn wiederum Informationen mit anderen Informationen verknüpft werden, entsteht auf einer noch höheren Ebene der Begriffshierarchie Wissen (siehe Abbildung 2-9, vgl. REHÄUSER U. KRCMAR 1996).

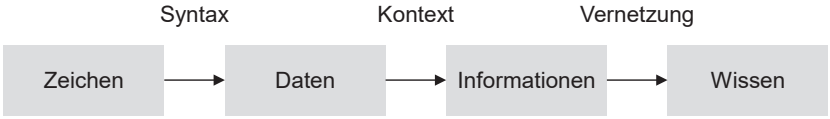

Abbildung 2-9: Begriffshierachie von Zeichen, Daten, Informationen und Wissen (eigene Darstellung i. A. a. KRCMAR 2015, S. 12)

Im Rahmen dieser Arbeit wird der Fokus insbesondere auf den Übergang von Daten zu Informationen gelegt, da dies eines der Hauptziele von Anwendungen ist, die in Unternehmen entwickelt werden. Für die Entwicklung von Anwendungen ist ein Datenmanagement notwendig (s. BIETHAN U. ROHRIG 1990, S. 740; MEIER 1994, S. 456). Das primäre Ziel des Datenmanagements besteht darin, Unternehmensdaten effizient bereitzustellen und nutzbar zu machen (s. KRCMAR 2015). Eine entscheidende Anforderung bei der Anwendungsentwicklung ist die Nutzung von Daten, die den Compliance-Vorschriften des Unternehmens entsprechen (vgl. Kapitel 2.3.3). Zentrale Sicherheitsaspekte, wie Betriebs- und Ausfallsicherheit sowie Schutz vor Angriffen, stehen dabei im Vordergrund (s. KEMPER U. EICKLER 2009). Die Klassifizierung von Daten kann dabei unterstützen, Datenschutz und Sicherheit der Daten zu gewährleisten, indem sie eine sinnvolle Einteilung der Daten vornimmt (s. HENNESSY ET AL. 2009).

Die Klassifikation von Daten erfolgt nach spezifischen Kriterien und zielt darauf ab, Daten mit ähnlichen Merkmalen in verschiedene Gruppen zu organisieren (s. MAHANTI 2021, S. 28). Diese Einteilung fördert eine effizientere Datenverwaltung hinsichtlich Speicherung, Schutz, Abruf, Verteilung und Nutzung. Abbildung 2-10 gibt eine Übersicht darüber, wie Daten klassifiziert werden können.

Abbildung 2-10: Klassifizierungsmöglichkeiten von Daten (MAHANTI 2021, S. 28)

In dieser Arbeit wird speziell untersucht, welche organisatorischen Auswirkungen die Entwicklung und Implementierung von Low-Code-Anwendung auf Organisationen haben. Die Klassifizierung nach Datenkritikalität und Datensensitivität ermöglicht es Organisationen, einen maßgeschneiderten Ansatz zum Schutz ihrer Daten zu entwickeln. Dies ist insbesondere relevant, da Low-Code-Plattformen eine schnellere und flexiblere Entwicklung und Bereitstellung von Anwendungen ermöglichen, was wiederum spezifische Anforderungen an den Umgang mit Datenschutz und Datensicherheit stellt. Daher werden die beiden Klassifikationen nachfolgend näher erläutert.

Datenkritikalität

Datenkritikalität spiegelt wider, wie wichtig Daten für die Mission, Funktionen und Prozesse einer Organisation in Bezug auf Integrität und Verfügbarkeit sind (s. MAHANTI 2021, S. 34). Die Integrität und Verfügbarkeit von Daten sind zwei der drei Ziele von Informationssicherheit (siehe Kapitel 2.3.4). Die Kritikalität der Daten kann in drei Kategorien unterteilt werden (siehe Abbildung 2-11).

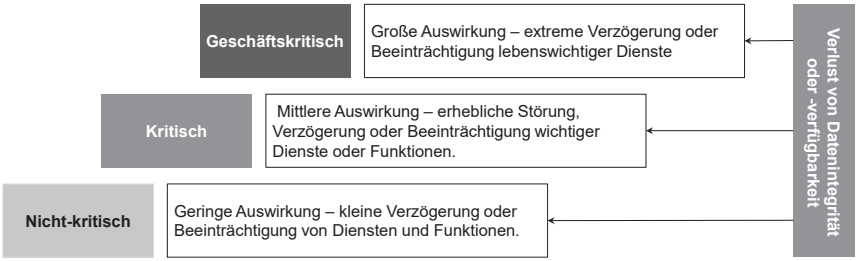

Abbildung 2-11: Datenklassifizierung nach Datenkritikalität (s. MAHANTI 2021, S. 34)

- **Nicht-kritische Daten**: Nicht-kritische Daten sind solche, bei deren Verlust an Integrität oder Verfügbarkeit nur geringfügige Verzögerungen oder Beeinträchtigungen von Diensten und Funktionen entstehen würden. Obwohl nicht-kritische Daten für eine effiziente Arbeitsweise einer Organisation erforderlich sind, hätte ein Verlust ihrer Integrität oder Verfügbarkeit nur minimale kurzfristige Auswirkungen auf die Geschäftskontinuität oder die betriebliche Effektivität.
- **Kritische Daten:** Kritische Daten sind entscheidend für die effiziente Arbeitsweise einer Organisation. Der Verlust der Integrität oder Verfügbarkeit kritischer Daten kann zu erheblichen Unterbrechungen, Verzögerungen oder Beeinträchtigungen lebenswichtiger Dienste oder Funktionen führen und hätte im Allgemeinen moderate kurzfristige Auswirkungen auf die Geschäftskontinuität oder betriebliche Effizienz.
- **Geschäftskritische Daten**: Geschäftskritische Daten sind für eine Organisation lebenswichtig, um effizient arbeiten zu können. Ein Verlust der Integrität oder Verfügbarkeit dieser unternehmenskritischen Daten würde zu einer extremen Verzögerung oder Beeinträchtigung lebenswichtiger Dienste oder Funktionen führen, sodass diese möglicherweise nicht erbracht werden könnten. Ein solcher Verlust hätte erhebliche kurzfristige Auswirkungen und könnte sogar

langfristige Konsequenzen für die Geschäftskontinuität oder betriebliche Effizienz haben. Der langfristige Verlust von unternehmenskritischen Daten könnte die Fähigkeit einer Organisation zur Wiederherstellung ernsthaft gefährden.

Datensicherheit

Die Klassifizierung von Daten aus der Perspektive der Datensicherheit basiert auf dem Grad ihrer Sensitivität und den Auswirkungen auf die Organisation, wenn die Daten ohne Autorisierung offengelegt, verändert, abgerufen, transportiert, gestohlen oder zerstört werden, wie in Abbildung 2-12 dargestellt. Je sensibler die Daten sind, desto höher sind der erforderliche Schutz, die Sicherheit und die Zugangskontrollen. MAHANTI definiert die Daten wie folgt:

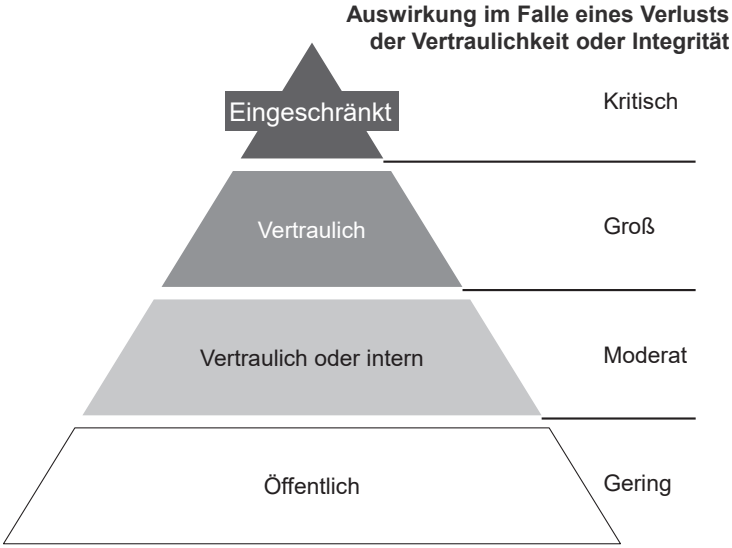

Abbildung 2-12: Datenklassifizierung nach Datensensibilität und -schutz (s. MAHANTI 2021, S. 36)

- **Eingeschränkte Daten**: Die Klassifikation von Daten als „eingeschränkt" erfolgt dann, wenn eine unbefugte Offenlegung, Änderung oder Zerstörung dieser Daten ein hohes Risiko mit sich bringt. Eingeschränkte Daten sind besonders sensible und äußerst vertrauliche Daten. Sie erfordern das höchste Maß an Datenschutz und Sicherheit bei Speicherung, Zugriff und Übertragung. Daher sollten für eingeschränkte Daten die strengsten Sicherheitsmaßnahmen, Zugangskontrollen und Schutzmaßnahmen angewendet werden. Daten, die durch globale, lokale, nationale Datenschutzbestimmungen und Vertraulichkeitsvereinbarungen geschützt sind, fallen in die Kategorie eingeschränkter Daten.
- **Vertrauliche oder interne Daten**: Vertrauliche oder interne Daten sind mäßig sensible Informationen, die vor unbefugtem Zugriff geschützt werden müssen und nur für einen begrenzten Personenkreis bestimmt sind. Der Unterschied

zwischen eingeschränkten Daten und vertraulichen Daten besteht darin, dass die Wahrscheinlichkeit, die Aussicht, die Dauer und das Ausmaß des Schadens im Falle vertraulicher oder interner Daten geringer sind als bei eingeschränkten Daten. Um unbefugten Zugriff zu verhindern, sind hoher Datenschutz, Zugriffsbeschränkungen und Sicherheitskontrollen erforderlich. Die Strenge ist jedoch geringer als bei eingeschränkten Daten.

- **Interne Daten**: Daten sollten als intern klassifiziert werden, wenn die unbefugte Offenlegung, Veränderung oder Zerstörung dieser Daten ein mäßiges Risiko für die Organisation darstellen könnte. Daten, die nicht ausdrücklich als eingeschränkte, vertrauliche oder öffentliche Daten klassifiziert sind, sollten standardmäßig als private oder interne Daten behandelt werden. Interne Daten sind für den internen Gebrauch in der Organisation bestimmt und nicht für die Veröffentlichung vorgesehen. Sie sollten auf kostengünstige Weise Zugriffs- und Sicherheitskontrollen unterliegen, obwohl der Aufwand bei weitem geringer ist als bei vertraulichen Daten.
- **Öffentliche Daten**: Daten sollten als öffentlich klassifiziert werden, wenn die unbefugte Offenlegung, Verteilung, Änderung, Nutzung oder Zerstörung dieser Daten für die Organisation ein vernachlässigbares oder kein Risiko darstellen und nur minimale nachteilige Auswirkungen haben würde. Öffentliche Daten sind die am wenigsten sensiblen Daten, haben die geringsten Sicherheitsanforderungen und können ohne Einschränkungen freigegeben werden. Es stellt sich jedoch immer die Frage, wer die Daten freigibt. Es sind nur geringe oder gar keine Kontrollen erforderlich, um die Privatsphäre öffentlicher Daten zu schützen. Es ist jedoch erforderlich, deren unbefugte Freigabe, Änderung oder Zerstörung zu verhindern.

Der Datenbegriff, wie in Abbildung 2-9 dargestellt und zuvor erläutert, wird in dieser Dissertation zugrunde gelegt. Zudem findet die Klassifizierung der Daten nach ihrer Kritikalität und Sensitivität bei der Entwicklung der Modelle Berücksichtigung (vgl. Kapitel 5.2.1 und Kapitel 6.2.5). Zusätzliche, in den nachfolgenden Kapiteln eingeführte Begriffe, werden klar in diesen Rahmen eingeordnet.

2.2.2 Informationssysteme und Anwendungen

Nachdem zuvor Zeichen, Daten, Informationen und Wissen voneinander abgegrenzt wurden, soll nun der Begriff des Informationssystems festgelegt werden. Im Folgenden werden die drei Begriffe Informationssystem, IT-System und Anwendung erläutert und für diese Arbeit definiert.

Informationssysteme sind soziotechnische Systeme, die menschliche und maschinelle Komponenten (Teilsysteme) umfassen (s. KRCMAR 2015, S. 23). Die ISO 42010 betrachtet ein Informationssystem als Systemelement, das mehrere IT-Systeme integriert. IT-Systeme sind spezifische, eigenständige Produkte, während das Informationssystem die höhere Ebene ist, die diese IT-Systeme koordiniert (s. ISO 42010). IT-Systeme sind spezifische, eigenständige Produkte, die sowohl physischen Hardware

Komponenten als auch softwareseitige Anwendungen beinhalten (s. KRCMAR 2015, S. 22). Eine Anwendung kann nach ZACHMAN als eine Gruppe von Funktionen betrachtet werden, während ein Informationssystem die Integration von Funktionen (Anwendungen) und Daten auf organisatorischer Ebene darstellt (s. ZACHMAN 1987).

In dieser Arbeit wird ein Informationssystem als sozio-technisches System mit menschlichen und technischen Komponenten verstanden. Die technischen Komponenten, welche hier als IT-Systeme definiert sind, unterteilen sich nochmal in die physischen Hardware-Komponenten und die softwareseitigen Anwendungen. Die Anwendung besteht aus Daten, die mit Funktionen verarbeitet und über Schnittstellen anderen Systemelementen aus dem Informationssystem zur Verfügung gestellt werden (siehe. Abbildung 2-13).

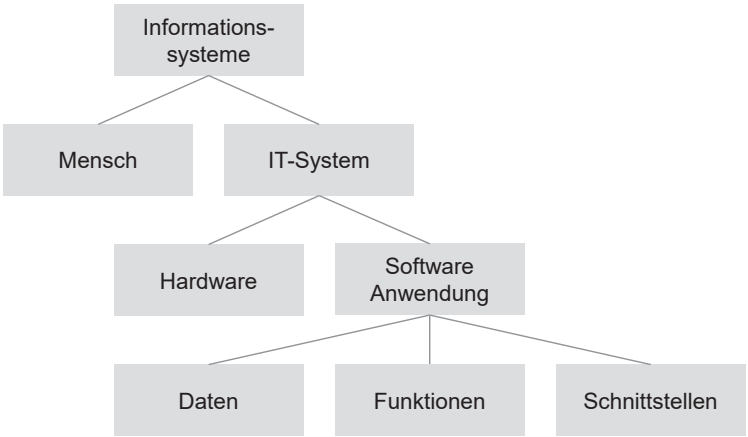

Abbildung 2-13: Komponenten eines Informationssystems (eigene Darstellung i. A. a. KRCMAR 2015, S. 22)

In der Literatur wird häufig zwischen langlebigeren und kurzfristigeren Anwendungen unterschieden. „McFarlan's Strategic Grid" unterscheidet zwischen operativen Anwendungen und strategischen Informationssystemen. Operative Anwendungen sind spezialisierte Systeme, die bestimmte Funktionen unterstützen, während strategische Informationssysteme auf eine strategische Ebene gehobene Ressourcen sind, die mehrere Anwendungen integrieren und das Unternehmen bei der strategischen Ausrichtung unterstützen. (s. MCFARLAN ET AL. 1983)

Auch im Aachener Digital Architecture Model (ADAM) wird zwischen Kernsystemen und Anwendungen unterschieden. Bei den Kernsystemen handelt es sich um statische, sich langsam ändernde Systeme, die das informationstechnologische Rückgrat eines Unternehmens bilden und zentrale Wertschöpfungsprozesse steuern sowie unterstützen (s. SCHUH ET AL. 2020). Anwendungen sind hingegen nutzerzentriert und erlauben es einem Nutzer, auf einfache und intuitive Art mit der Unternehmensressource „Information" wertstiftend zu agieren. Die Anwendungen unterliegen durch schnell wechselnde Veränderungen einem stetigen Wandel und sind durch agile und

flexible Anpassbarkeit geprägt (s. SCHUH ET AL. 2020). Gartner schlägt im Pace-Layer-Modell vor, Anwendungen basierend auf der Änderungsgeschwindigkeit in drei Schichten einzuteilen. „System of Record“ (Anwendungen mit stabilem und langsamem Wandel), „System of Differentiation“ (Anwendungen, die sich schneller ändern) und „System of Innovation“ (Anwendungen, die schnellen Wandel und Innovationen ermöglichen) (s. MESAGLIO U. HOTLE 2016).

In dieser Arbeit werden Anwendungen als Komponenten des Informationssystems verstanden und können sowohl für einen langfristigen, strategischen Zweck als auch für kurzlebigere, nutzerzentrierte Einsätze dienen (siehe Kapitel 5.2.1).

Ferner kann man bei der Bereitstellung von Anwendungen grundsätzlich zwischen Beschaffung von Standardsoftware und Erstellung von Individualsoftware unterscheiden. Abbildung 2-14 zeigt die verschiedenen Alternativen.

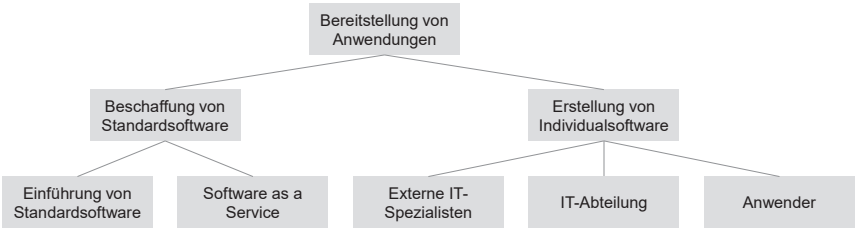

Abbildung 2-14: Alternativen der Softwarebereitstellung (s. KRCMAR 2015, S. 201-202)

Der Erwerb von betrieblicher Standardsoftware kann in „Standardsoftware“ und „Software as a Service“ unterteilt werden. Bei der „Einführung von Standardsoftware“ erwirbt das Unternehmen die Software zu einem bestimmten Versionsstand und darf sie im Rahmen des Lizenzvertrags nutzen (s. KRCMAR 2015, S. 2012). Für „Software as a Service“ bezahlt das Unternehmen einen regelmäßig wiederkehrenden Betrag und erhält zusätzlich zum Nutzungsrecht einen Service der Wartung und Weiterentwicklung durch den Anbieter. Bei der Erstellung von Individualsoftware gibt es drei Umsetzungskonzepte: Externe IT-Spezialist*innen können die Individualsoftware entwickeln, oder die interne IT-Abteilung des Unternehmens verfügt über ausreichende Ressourcen für die Umsetzung des Softwareprojekts. Eine besondere Rolle spielt die Erstellung von Individualsoftware durch die Anwender*innen, bei dem erfahrene Hauptanwender*innen (Power-User) oder konventionelle Endanwender*innen (End-User) die Möglichkeit haben, die Entwicklung von Anwendungen zu übernehmen.

Der Fokus dieser Arbeit liegt auf der Erstellung von Individualsoftware mithilfe von Low-Code-Plattformen. Die Anwendungen können sowohl von der IT-Abteilung als auch durch die Endanwender*innen entwickelt werden. Dabei kann ein Unternehmen externe IT-Spezialist*innen zur Unterstützung hinzuziehen. Die Auswahl einer Low-Code-Plattform, wird explizit nicht berücksichtigt. Es wird davon ausgegangen, dass das Unternehmen sich bereits für eine Low-Code-Plattform entschieden hat. Die Low-Code-Plattform dient dabei als Werkzeug, welches Funktionen, Daten und Schnittstellen verwaltet, um die Anwendungsentwicklung zu ermöglichen.

2.2.3 Integration und Schnittstellen

Die Integration von Anwendungen (siehe Kapitel 2.2.2) zur gemeinsamen Datennutzung entlang der Wertschöpfungskette ist eine wichtige Grundlage für die Umsetzung der Industrie 4.0 (s. HOFFMANN 2018, S. 23; SCHUH ET AL. 2017, S. 28). Unter Integration wird hierbei das Zusammenfügen von Einzelteilen zu einer Einheit unter Anwendung einer bestimmten Methode verstanden (s. KAIB 2002, S. 10). Im Rahmen dieser Arbeit bezieht sich die Integration auf das Zusammenfügen verschiedener Elemente innerhalb eines Informationssystems (siehe Kapitel 2.2.2). DERN differenziert in seiner Analyse vier wesentliche Dimensionen der Integration (siehe Abbildung 2-10):

- Der **Integrationsgegenstand** definiert die spezifischen Elemente innerhalb eines Systems, die integriert werden sollen.
- Die **Integrationsreichweite** legt fest, ob die Integration innerhalb eines Unternehmens oder zwischen verschiedenen Unternehmen stattfindet.
- Der **Automatisierungsgrad** bestimmt, in welchem Ausmaß menschliches Eingreifen bei der Integration erforderlich ist.
- Die **Integrationsperspektive** bestimmt die grundlegende Sichtweise auf die Integration. Dabei wird beispielsweise zwischen betriebswirtschaftlich, organisatorisch und technisch unterschieden.

Abbildung 2-15 zeigt die vier Dimensionen der Integration und den Bezug zu dieser Arbeit.

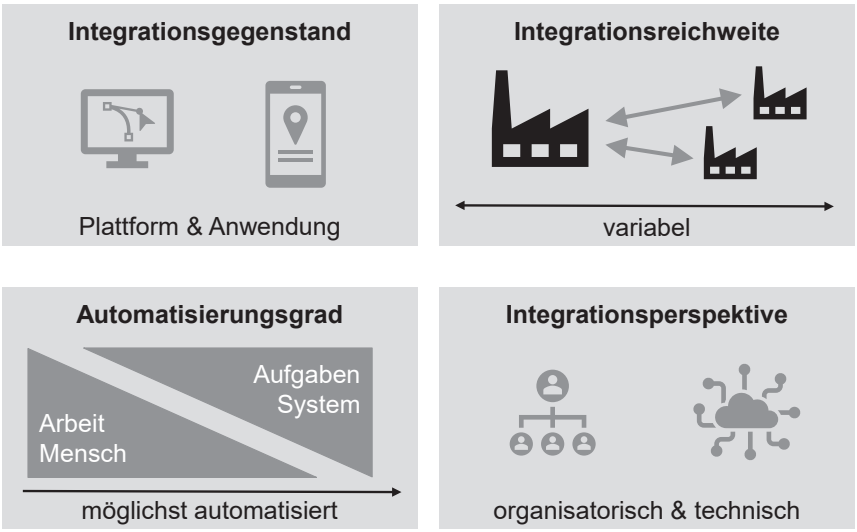

Abbildung 2-15: Dimensionen des Integrationsmanagements (eigene Darstellung i. A. a. DERN 2011, S. 41)

Im Rahmen dieser Arbeit sind die Integrationsgegenstände einmalig die Low-Code-Plattform und anschließend kontinuierlich die Low-Code-Anwendung, die nachhaltig in

ein bestehendes Informationssystem integriert werden sollen. Die Integrationsreichweite kann je nach Anwendungsfall variieren (siehe Kapitel 5.2.1). Beim Automatisierungsgrad wird aus Effizienzgründen ein möglichst hoher Automatisierungsgrad angestrebt. Bei Betrachtung der Integrationsperspektive wird die betriebswirtschaftliche Sichtweise in dieser Arbeit außer Acht gelassen. Stattdessen wird sich auf die organisatorische sowie die technische Sicht fokussiert. Während in Kapitel 2.3 auf die Bestandteile der organisatorischen Integration eingegangen wird, wird im Folgenden auf die technischen bzw. systemseitigen Möglichkeiten der Integration der verschiedenen Low-Code-Anwendung eingegangen.

Das Design externer Schnittstellen zwischen Anwendungen erfordert spezifische Informationen über die Anwendung, an die Daten gesendet oder von der Daten empfangen werden. Die Art der Daten, die übermittelt werden, sollten bereits frühzeitig gesammelt und überprüft werden. Außerdem sollte das Design der Schnittstelle Fehlerprüfung und angemessene Sicherheitsfunktionen (siehe Kapitel 6.2.5) beinhalten (s. Pressman u. Maxim 2020, S. 175).

Weiterhin unterscheidet man zwischen zwei Arten der Vernetzung von Systemen: Proprietäre und neutrale Schnittstellen. Eine proprietäre Schnittstelle verwendet ein einzigartiges Format zur Übertragung von Informationen, das ausschließlich für die spezifische Anwendung oder Schnittstelle des jeweiligen Anbieters entwickelt wurde. Daher ist es notwendig, für jede unterschiedliche Kombination von Anwendungen eine neue und individuell angepasste Schnittstelle zu schaffen. Im Gegensatz dazu verwendet eine neutrale Schnittstelle ein offenes Format, das branchenweit oder für eine bestimmte Kategorie von Anwendungen als Standard gilt (z. B. REST API). Dies ermöglicht es Anwendungen verschiedener Anbieter, Informationen über dieselbe Schnittstelle auszutauschen. Dadurch ist es ausreichend, für jede Anwendung lediglich eine einzige neutrale Schnittstelle zu entwickeln. (s. Eigner et al. 2014, S. 328-329)

Abbildung 2-16 zeigt den Unterschied zwischen proprietären und neutralen Schnittstellen.

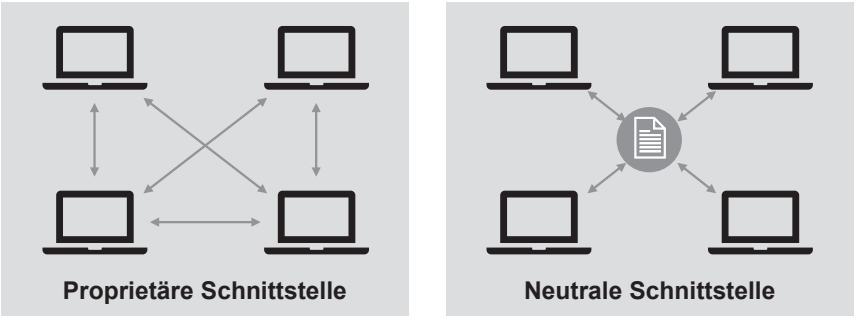

Abbildung 2-16: Proprietäre und neutrale Schnittstellen (s. Eigner et al. 2014, S. 328-329)

Eine besondere Art der Schnittstellen sind service-basierte Schnittstellen, auch „Application Programming Interface“ (API) genannt. APIs sind Schnittstellen, die Entwickler*innen den Zugriff auf die Daten und Funktionen bzw. den Service einer anderen Anwendung ermöglichen (s. SAWANT U. BACCHELLI 2017). Das bedeutet, dass der Service direkt aufgerufen werden kann und nicht nur über die Benutzeroberfläche der Anwendung zugänglich ist (s. SOMMERVILLE 2016, S. 759). Das ermöglicht es Entwickelnden die entsprechende Anwendung über Protokolle zu koppeln, statt den gesamten Quellcode des Services zu übernehmen (s. PRESSMAN U. MAXIM 2020, S. 273).

In dieser Arbeit liegt der Fokus auf den Schnittstellen zwischen der Low-Code-Plattform und den restlichen Anwendungen. Bei den meisten Low-Code-Plattformen werden diese Schnittstellen als APIs implementiert (s. LUO ET AL. 2021). Zur Beschreibung dieser Schnittstellen werden diese in ihrem Standardisierungsgrad unterschieden.

Es gibt eine Vielzahl von standardisierten Schnittstellen zwischen Anwendungen, die je nach Anwendungsgebiet variieren. Im Rahmen dieser Arbeit wird auf keine spezifischen Standards eingegangen, sondern vielmehr der Standardisierungsgrad verschiedener Schnittstellen definiert und im Kontext der Softwareentwicklung beleuchtet.

- **Standardisierte Schnittstellen**: Standardisierte Schnittstellen sind nach allgemein anerkannten Normen und Standards entwickelt. Diese Normen können von Industrieorganisationen oder Standardisierungsgremien festgelegt werden und dienen als Rahmen, der die Interoperabilität zwischen verschiedenen Systemen ermöglicht. Standardisierte Schnittstellen erleichtern die Integration und den Austausch von Daten zwischen unterschiedlichen Anwendungen und Plattformen, da sie auf allgemein akzeptierten Regeln und Spezifikationen basieren.
- **Konfigurierbare Schnittstellen**: Konfigurierbare Schnittstellen bieten Flexibilität in Bezug auf die Anpassung an spezifische Anforderungen oder Umgebungen. Diese Schnittstellen können anpassbare Parameter und Einstellungen aufweisen, die von Benutzern oder Administratoren konfiguriert werden können, um verschiedene Szenarien oder Anforderungen zu erfüllen. Die Konfigurierbarkeit ermöglicht eine gewisse Vielseitigkeit, ohne dass dabei grundlegende Strukturen geändert werden müssen.
- **Individuelle Schnittstellen**: Individuelle Schnittstellen werden spezifisch für die Anforderungen einer bestimmten Anwendung entwickelt. Diese Schnittstellen sind maßgeschneidert und auf die speziellen Bedürfnisse und Funktionen eines bestimmten Kontexts zugeschnitten. Obwohl individuelle Schnittstellen eine präzise Passform bieten können, könnten sie auch zu Inkompatibilitäten führen, wenn sie nicht mit anderen Anwendungen oder Standards übereinstimmen.

In einem allgemeinen Kontext bieten standardisierte Schnittstellen die breiteste Interoperabilität. Konfigurierbare Schnittstellen sind anpassbar, um bestimmte Anforderungen zu erfüllen. Individuelle Schnittstellen werden für spezifische Zwecke entwickelt. Die Wahl zwischen diesen Ansätzen hängt von den spezifischen Anforderungen, Flexibilitätsbedürfnissen und dem gewünschten Maß an Interoperabilität ab. Eine API

kann sowohl als standardisierte, konfigurierbare als auch als individuelle Schnittstelle dienen.

2.2.4 Informationssystem-Architektur und IT-Architektur

In den vorherigen Kapiteln wurden die Bausteine einer IT-Architektur beschrieben. Generell versteht man unter einer Architektur die Anordnung und Verbindung von Systemelementen (s. ISO 15704, S. 1; FRIEDLI 2000, S. 79). Unternehmensleitungen und IT-Verantwortliche stehen oft vor der Herausforderung, aus der Vielzahl an Anforderungen und IT-Lösungen eine passende Architektur zu wählen. In der Vergangenheit führte dies zu unkoordinierten IT-Systemen und Applikationen, die im Betrieb und Support Probleme verursachten (s. TIEMEYER 2020, S. 86).

In der Fachliteratur existieren unterschiedliche Definitionen für die Begriffe IT-Architektur und Informationssystem-Architektur. Daher wird für diese Arbeit eine spezifische Definition dieser Begriffe auf Basis der bereits in Kapitel 2.2.2 definierten Definitionen für Informationssysteme und Anwendungen festgelegt. Abbildung 2-17 illustriert die Unterscheidung zwischen Informationssystem-Architektur und IT-Architektur.

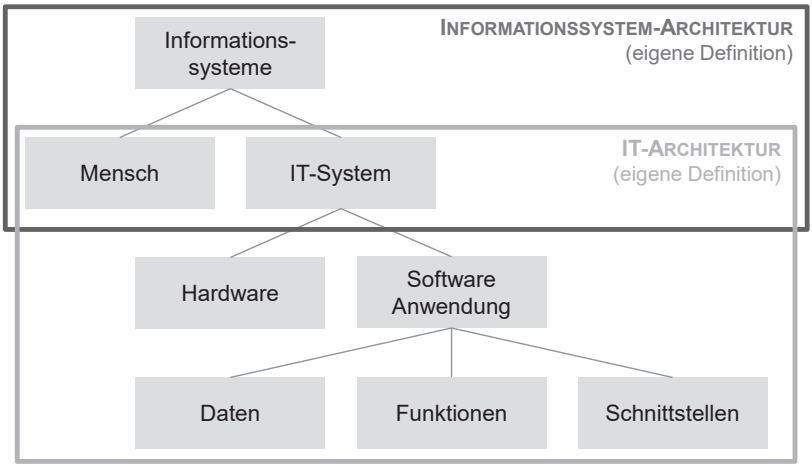

Abbildung 2-17: Abgrenzung Informationssystem-Architektur und IT-Architektur (eigene Darstellung)

In dieser Dissertation wird die Informationssystem-Architektur so definiert, dass sie sich spezifisch auf die IT-Systeme und die Interaktion der Menschen mit diesen Systemen fokussiert. Demgegenüber ist die IT-Architektur in dieser Dissertation ein umfassenderes Konzept, das die gesamte technische Infrastruktur einer Organisation umfasst. Während verschiedene Modelle der Informationssystem-Architektur in Kapitel 3.2.2 diskutiert werden, wird der Begriff der IT-Architektur im Folgenden näher erläutert.

Für die Darstellung von IT-Architekturen werden verschiedene Teilarchitekturen herangezogen (s. TIEMEYER 2020, S. 45). Abbildung 2-18 gibt einen Überblick über die verschiedenen Bereiche der IT-Architektur.

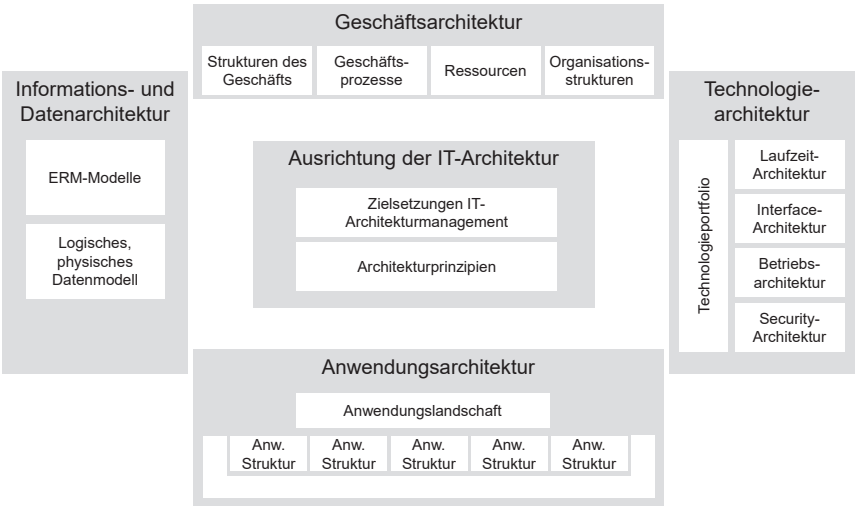

Abbildung 2-18: Bereiche der IT-Architektur im Überblick (s. TIEMEYER 2020, S. 45)

Eine der Hauptaufgaben des IT-Managements ist es, auf der Grundlage strategischer Überlegungen und bestehender Anforderungen einen Wegweiser für die zukünftige Entwicklung der IT-Landschaft zu erstellen. Dies schließt die Definition einer Zielarchitektur für die IT-Infrastruktur und die Anwendungslandschaft ein. Dafür sind Richtlinien für die Gestaltung und Entscheidungen bezüglich der Systeme zu formulieren, ebenso wie die Auswahl von „strategischen" Technologien und Produkten, die als Standards innerhalb der Organisation festgelegt und kommuniziert werden müssen. Zur Sicherstellung einer effektiven strategischen Ausrichtung sind verbindliche „Roadmaps" für die Weiterentwicklung der IT, einschließlich der Erarbeitung spezifischer Architekturvorgaben und -standards, zu entwickeln. (s. TIEMEYER 2020, S. 88)

Diese Arbeit fokussiert sich auf die Auswirkungen, die die Einführung von Low-Code-Plattformen auf die IT-Architektur hat. Untersucht wird, wie durch den Einsatz von Low-Code die Applikationsarchitektur erweitert werden kann (siehe Kapitel 5) und wie Daten innerhalb der Informations- und Datenarchitektur verwaltet und geschützt werden können. Die Implementierung einer Low-Code-Plattform wird dabei als technische Bereicherung der Technologiearchitektur betrachtet.

Zudem führt die Einführung von Low-Code nicht nur zu technischen, sondern auch zu organisatorischen Veränderungen. Das Management der IT-Architektur bietet eine umfassende Sicht auf alle Elemente der IT-Landschaft und deren Verknüpfung mit den unterstützten Geschäftsprozessen. Diese Betrachtungsweise geht über die reine Software- und Hardware-Ebene hinaus und sieht die IT als integralen Bestandteil des

Geschäftsbetriebs und der organisatorischen Strukturen. Diese Ausweitung mündet in das Konzept des Enterprise Architecture Managements (EAM) (s. TIEMEYER 2020, S. 87).

2.3 Management von IT-Architekturen

Nachdem im vorherigen Kapitel die technischen Konzepte und Begriffe zur Implementierung und zum Betrieb von Low-Code-Anwendungen dargelegt wurden, konzentriert sich dieses Kapitel auf die organisatorischen Aspekte. Zunächst wird der Begriff des Enterprise Architecture Managements erläutert, einschließlich des darin enthaltenen „IT-Governance-Risk-Compliance-Trias“ (IT-GRC-Trias, siehe Kapitel 2.3.1). In den Kapiteln 2.3.2 bis 2.3.4 erfolgt eine vertiefte Betrachtung des IT-GRC-Trias, wobei IT-Governance, IT-Compliance und IT-Risikomanagement jeweils einzeln beschrieben und im Kontext dieser Arbeit analysiert werden. Abschließend wird die Informationssicherheit, als ein signifikantes IT-Risiko im Zusammenhang mit dem Einsatz von Low-Code-Plattformen, explizit behandelt (siehe Kapitel 2.3.5).

2.3.1 Enterprise Architecture Management

Enterprise Architecture Management (EAM) ist ein ganzheitlicher Ansatz zur Planung und Steuerung der gesamten Unternehmensarchitektur mit dem Ziel, die IT-Architektur auf die Unternehmensvision und -strategie auszulegen. Es umfasst die Schaffung, Kommunikation und Verbesserung von Schlüsselprinzipien und -modellen, die den zukünftigen Zustand des Unternehmens aus der Perspektive der Unternehmensarchitektur darstellen und seine Entwicklung unterstützen (s. HANSCHKE 2016, S. 8). EAM fördert die Transparenz der IT-Landschaft und unterstützt die Geschäftsprozesse, was die Grundlage für die Maßnahmen zur Konsolidierung und Komplexitätsreduktion bildet (s. HANSCHKE 2016, S. 10). Dabei umfasst die Geschäftsarchitektur primär die Basisstrukturen des Geschäfts einschließlich der Ziele, der zugrunde liegenden Geschäftsprozesse sowie der erforderlichen Steuerungsmechanismen und organisatorischen Anpassungen (s. TIEMEYER 2020, S. 138). Die Ziele von EAM variieren von operativ bis strategisch und sind abhängig von der Ausgangslage, den Rahmenbedingungen und der IT-Positionierung (s. HANSCHKE 2016, S. 194). Abbildung 2-19 zeigt die unterschiedlichen Schwerpunkte, die gesetzt werden können.

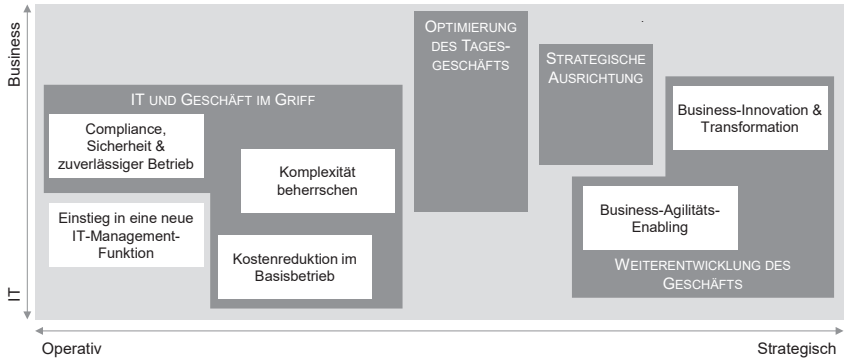

Abbildung 2-19: Herausforderungen für Verantwortliche in Business und IT (i. A. a. Hanschke 2016, S. 194)

Diese Arbeit konzentriert sich auf die operative Unterstützung bei der Implementierung und dem Betrieb von Low-Code-Anwendung in Unternehmen. Der Fokus liegt auf der Integration von Low-Code-Anwendung in die vorhandene IT-Architektur und Organisationsstruktur. Dabei soll der Fokus insbesondere auf die Compliance, Sicherheit und den zuverlässigen Betrieb gelegt werden. Die damit einhergehende Komplexität zu beherrschen, ist eine Herausforderung, die durch EAM verwaltet werden kann, indem EAM eine strukturierte und ganzheitliche Sicht auf die Prozesse, Informationssysteme, Technologien und Infrastrukturen eines Unternehmens schafft. Allerdings werden im Rahmen des EAMs primär grobgranulare Informationen berücksichtigt, die einen Überblick über Risiken bieten und Compliance-relevante Elemente identifizieren (s. Hanschke 2016, S. 205).

Die mit der Einführung von Low-Code-Plattformen verbundenen neuartigen Chancen und Risiken werden zunächst nur oberflächlich behandelt. Eine detaillierte Analyse dieser Chancen und Risiken erfordert eine fokussierte Betrachtung der strategischen Ausrichtung des EAMs und der operativen Betriebsführung. In diesem Zusammenhang hat sich der Begriff IT-GRC-Management als Bezeichnung für die integrierte Steuerung der Bereiche IT-Governance, IT-Risikomanagement und IT-Compliance etabliert (s. Knoll u. Strahringer 2017, S. 2). Die wachsende Bedeutung und die engen inhaltlichen Verknüpfungen dieser drei Bereiche bedingen die Entwicklung der „Governance-Risk-Compliance“ (GRC)-Trias, welche eine integrierte Strategie und ein einheitliches Management erfordert (s. Klotz u. Dorn 2008).

IT-Compliance-Strategien, -Prozesse, -Verantwortlichkeiten, -Techniken und -Instrumente müssen sich daher in die allgemeinen Compliance-Aktivitäten eines Unternehmens, die sogenannte „Corporate Compliance“, einfügen und zugleich Teil der GRC-Trias werden. Darüber hinaus gibt es spezifische IT-bezogene Ausprägungen innerhalb der Corporate Governance und des Risk Managements, nämlich IT-Governance und IT-Risk Management. Dies führt zur Entstehung einer speziellen „IT-GRC-Trias“,

die in die umfassendere „Corporate GRC“ integriert werden muss, um eine ganzheitliche Steuerung und Überwachung aller Governance-, Risiko- und Compliance-Aspekte sicherzustellen (s. KLOTZ U. DORN 2008). Abbildung 2-20 zeigt, wie die drei Bereiche zusammenhängen und welchen Einfluss die Implementierung und der Betrieb von Low-Code-Anwendung nimmt.

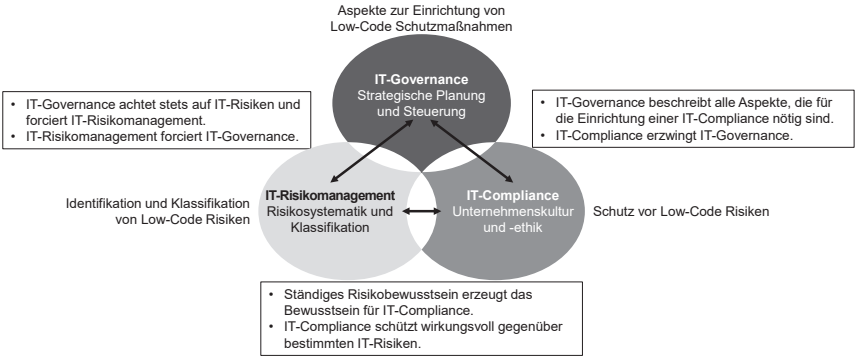

Abbildung 2-20: Das IT-GRC-Dreieck (s. KLOTZ U. DORN 2008)

Die durch Low-Code hinzugekommenen Risiken erfordern eine IT-Compliance, die gegen diese Risiken schützt. Die IT-Governance liefert dazu die richtigen Aspekte, um Schutzmaßnahmen einzuführen. Die Implementierung und der Betrieb von Low-Code-Anwendung erfordert nicht nur technische Überlegungen, sondern auch ein umfassendes Verständnis von IT-Governance, IT-Compliance und IT-Risikomanagement. Ihre Rolle und Bedeutung für diese Arbeit werden im Folgenden detailliert erläutert.

2.3.2 IT-Governance

Governance beschreibt grundsätzlich die verantwortungsvolle, transparente und nachvollziehbare Führung und Überwachung von Organisationen mit Ausrichtung an Regulierungen, Standards und ethischen Grundsätzen (s. JOHANNSEN U. GOEKEN 2011). In der Wirtschaft hat sich der Terminus „Corporate Governance“ etabliert, der spezifisch auf die Leitung und Überwachung von Unternehmen aus einer ganzheitlichen Perspektive unter Berücksichtigung dieser Governance-Prinzipien abzielt (s. KNOLL U. STRAHRINGER 2017). Für die DACH-Region ist die Corporate Governance wesentlich durch den Corporate Governance Kodex von 2019 geprägt, der das Ziel verfolgt, eine gute und verantwortungsvolle Unternehmensführung zu sichern, wobei ein entsprechendes Qualitätsniveau bei Unternehmensleitungen und deren Überwachung angestrebt wird (s. BERWANDER U. HAHN 2020).

Corporate Governance gewährleistet, dass die Bedürfnisse und Optionen der Stakeholder angemessen berücksichtigt werden, um ausgeglichene und konsensbasierte Unternehmensziele festzulegen (s. ISACA 2020). Die ISO 38500, die sich auf die Corporate Governance der Informationstechnologie konzentriert, verdeutlicht die Verbindung zwischen Corporate Governance und IT-Governance. Da die Corporate

Governance alle Unternehmensprozesse umfasst, um eine verantwortungsvolle Unternehmensführung und -kontrolle zu garantieren, umfasst sie auch die zunehmend wichtiger werdenden IT-Unterstützungsprozesse (s. KNOLL U. STRAHRINGER 2017). Dadurch werden wesentliche Steuerungs- und Regelungsfunktionen im Bereich der IT-Governance definiert, die die fachlichen Aspekte der Informationstechnologie in den Fokus rücken. IT-Governance ist demnach ein Teilbereich der Corporate Governance, wobei in der Literatur keine einheitliche Definition existiert (s. KNOLL U. STRAHRINGER 2017). Einige Autor*innen beschreiben IT-Governance als eine Sammlung von Prinzipien, Instrumenten oder Maßnahmen (vgl. MEYER ET AL. 2003, S. 445; TIEMEYER 2020, S. 465-446), während andere den Begriff „Rahmenwerk" verwenden (s. LACKES U. SIEPERMANN 2018). Allen Definitionen ist dennoch gemein, dass die IT-Governance den effektiven Einsatz von IT zur Unterstützung der Unternehmensziele verfolgt. Durch den Einsatz von IT-Governance wird gewährleistet, dass die Prozesse, Organisationsstrukturen, Führungspraktiken und die gesamte IT-Infrastruktur im Unternehmen konsequent entwickelt, gesteuert und überwacht werden. Diese klare Strukturierung der Unternehmens-IT ermöglicht die Einhaltung von Vorschriften und Standards und trägt zur optimalen Funktionsweise bei. IT-Governance schafft daher Transparenz in der Unternehmens-IT und führt zur Harmonisierung der IT-Landschaft. Es ist wichtig zu beachten, dass IT-Governance kein Projekt ist, das einen festen Anfangs- und Endpunkt hat, sondern vielmehr ein grundlegendes Element in einer Organisation darstellt (s. MEYER ET AL. 2003, S. 445-446).

Abbildung 2-21 veranschaulicht die Verbindung zwischen Corporate Governance und IT-Governance. Corporate Governance berücksichtigt eine Vielzahl von Einflussfaktoren und leitet daraus konkrete Regeln und Richtlinien für den Betrieb des Unternehmens ab. Diese Regeln und Vorgaben haben auch Auswirkungen auf die Gestaltung und Funktionsweise der Unternehmens-IT. Daher wird IT-Governance etabliert, um detaillierte Leitlinien speziell für die Unternehmens-IT festzulegen (s. RÜTER ET AL. 2010, S. 5).

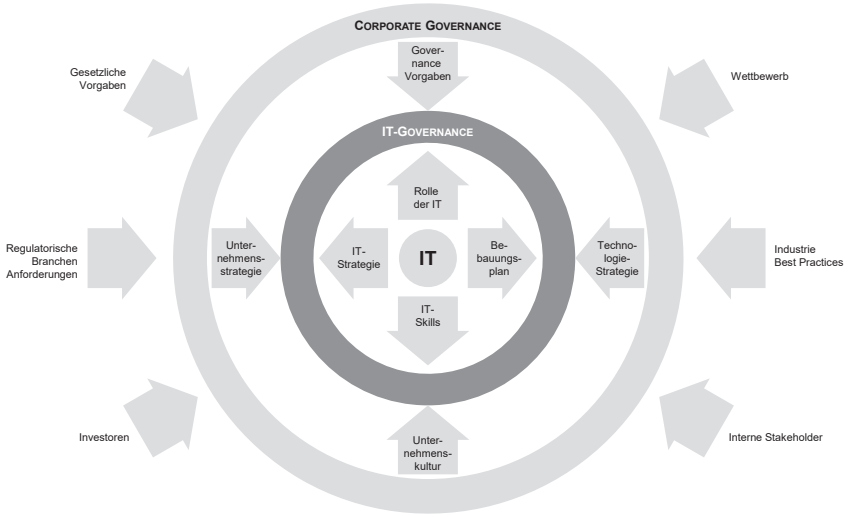

Abbildung 2-21: Bezugsrahmen IT-Governance (s. RÜTER ET AL. 2010, S. 19)

In dieser Arbeit wird untersucht, wie vorhandene IT-Governance-Strukturen in Unternehmen ausgebaut werden können, um die Risiken, die mit dem Einsatz von Low-Code-Plattformen einhergehen, effektiver zu managen und einen zuverlässigen Betrieb zu gewährleisten. Durch die Einbindung von Low-Code-Plattformen entstehen neue Herausforderungen für die IT-Governance. Die schnelle Entwicklung und Bereitstellung von Anwendungen durch Low-Code erfordert eine Anpassung der Governance-Strukturen, um sowohl die Agilität als auch die Sicherheit und Compliance zu gewährleisten. Dies schließt eine Überprüfung und gegebenenfalls Anpassung der bestehenden Kontrollmechanismen, Richtlinien und Standards ein, um die spezifischen Risiken von Low-Code-Plattformen angemessen zu adressieren und den sicheren und effizienten Einsatz dieser Technologie zu fördern.

Die IT-Governance konzentriert sich auf die Klärung von Verantwortlichkeiten, Entscheidungsrechten sowie die dazugehörigen Prozesse und Verfahren, um den maximalen Wertbeitrag aus der Nutzung von IT für das Unternehmen zu erzielen. Dazu gehört auch die Implementierung eines IT-Kontrollsystems als Teil des unternehmensweiten internen Kontrollsystems, das die Steuerung und Überwachung der IT im Unternehmen sicherstellt. Für die Konzeption des IT-Kontrollsystems gibt es einschlägige Standards, die im Bereich der IT-Compliance relevant sind (s. TIEMEYER 2020, S. 600-601).

2.3.3 IT-Compliance

Compliance bezieht sich auf ein Verhalten, das Regeln und ethischen Standards entspricht. In der Wirtschaft wird die Corporate Compliance als die Übereinstimmung eines Unternehmens mit relevanten Vorschriften verstanden (s. KLOTZ U. BARTONITZ

2023). Dies umfasst alle Maßnahmen, die die Einhaltung von Gesetzen, Richtlinien oder Unternehmenskodizes gewährleisten (s. SCHMIDT ET AL. 2021, S. 7). Basierend auf den Anforderungen der Corporate Compliance fokussiert sich die IT-Compliance auf die Einhaltung der Vorgaben, die alle technischen und organisatorischen Aspekte der Unternehmens-IT betreffen, wobei die Einhaltung sowohl von internen Abteilungen als auch von externen Dienstleistern gewährleistet sein muss (s. SCHMIDT ET AL. 2021, S. 9). IT-Compliance wird durch angemessene Maßnahmen erreicht und erfordert die eindeutige Nachweisbarkeit verbindlicher Rechtsnormen durch strukturierte Dokumentation (s. TIEMEYER 2020, S. 849), wie in Abbildung 2-22 dargestellt.

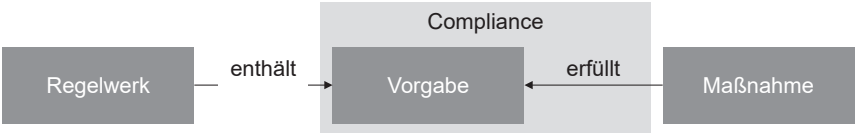

Abbildung 2-22: IT-Compliance als Erfüllung von Vorgaben (s. TIEMEYER 2020, S. 849)

Die Vorschriften schließen nicht nur Gesetze ein, sondern berücksichtigen auch andere Regelwerke, denen sich eine Organisation verpflichtet fühlt (s. KLOTZ U. BARTONITZ 2023). Dabei lässt sich zwischen drei verschiedenen Quellen von Compliance Vorgaben unterscheiden: Unternehmensinterne Regelwerke, Rechtliche Vorgaben und unternehmensexterne Regelwerke (siehe Abbildung 2-23).

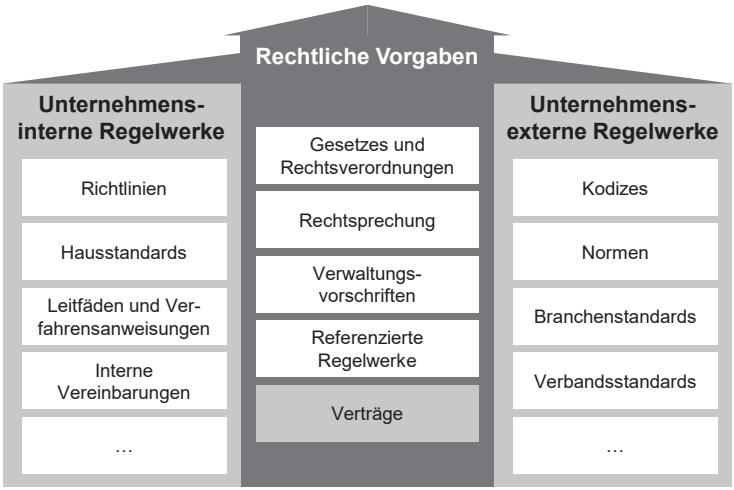

Abbildung 2-23: Quellen von Compliance-Vorgaben – das „House of Compliance" (s. TIEMEYER 2020, S. 867)

Da innerhalb dieser Arbeit ein unternehmensinternes Regelwerk entwickelt werden soll, werden zunächst die Begriffe Regeln und Regelwerk voneinander abgegrenzt und anschließend wird das unternehmensinterne Regelwerk basierend auf TIEMEYER detaillierter beschrieben.

Regeln enthalten spezifische Vorgaben durch deren Einhaltung IT-Compliance erreicht wird (s. TIEMEYER 2020, S. 849). Die Sammlung und Organisation dieser Regeln in einem systematischen Rahmen werden als Regelwerk verstanden. Bei unternehmensinternen Regelwerken liegt die Bindungswirkung ausschließlich beim Unternehmen, das diese Regelungen erlässt. Beispiele für solche Regelungen im IT-Bereich umfassen interne IT-Richtlinien oder Verfahrensvorgaben zur IT-Sicherheit, wie IT-Sicherheitsvorschriften. Auch zwischen der IT-Abteilung und den Fachabteilungen vereinbarte Service-Level-Agreements (SLAs) gehören zu dieser Kategorie.

Interne Regelwerke sind aus zwei Gründen für die Compliance relevant:

- **Sicherstellung der Einhaltung anderer Regelwerke**: Sie dienen oft dazu, die Beachtung der Anforderungen aller anderen Regelwerke zu gewährleisten, indem sie konkrete Handlungsanweisungen für Organisationsmitglieder vorgeben.
- **Dokumentation externer Verpflichtungen**: Sie dokumentieren gegenüber externen Parteien, insbesondere in Bezug auf rechtliche Vorgaben, dass das Unternehmen seinen Verpflichtungen nachkommt.

Diese internen Regelwerke tragen somit entscheidend dazu bei, dass ein Unternehmen sowohl intern als auch extern seinen Compliance-Verpflichtungen gerecht wird. Sie stellen sicher, dass die Organisation in Einklang mit gesetzlichen Bestimmungen handelt und gleichzeitig interne Prozesse und Verhaltensweisen klar definiert und geregelt sind (s. TIEMEYER 2020, S. 874-875).

Zusammenfassend wird IT-Compliance in dieser Arbeit als ein Zustand definiert, der durch die Einhaltung und Implementierung aller Regelwerke erreicht wird, die das Verhalten der Unternehmens-IT steuern und regulieren. Diese Regelwerke können gesetzlich vorgeschrieben oder vertraglich festgelegt sein oder auch freiwillig vom Unternehmen übernommen werden.

2.3.4 IT-Risikomanagement

Die Betrachtung des Risikobegriffs aus dem Finanzsektor, die sowohl positive als auch negative Konsequenzen eines zukünftigen Ereignisses einbezieht (KÖNIGS 2017; SEIBOLD 2006), ist auch in anderen Bereichen, wie der IT und der Compliance, von Bedeutung. Diese Dualität von Risiken und Chancen unterstreicht, dass Entscheidungen bewusst getroffen werden müssen, wobei potenzielle Gewinne gegen mögliche Schäden abgewogen werden. Insbesondere im Kontext der Digitalisierung können neue Möglichkeiten nur durch das bewusste Eingehen von Risiken erschlossen werden (s. KNOLL U. STRAHRINGER 2017, S. 4).

Im Hinblick auf Informationssysteme haben sich zwei Perspektiven entwickelt: das IT-Risiko, das eher technische Aspekte betont, und das Informationssicherheitsrisiko, das sich auf Inhalte und Geschäftsprozesse konzentriert (siehe Kapitel 2.3.5). Beide Ansätze sind valide und ergänzen sich gegenseitig, um ein umfassendes Verständnis von Risiken zu gewährleisten. Ein Risiko wird demnach als potenzielles, ungewisses

und plötzlich eintretendes Ereignis definiert, das negative Auswirkungen auf die Unternehmensziele, die Geschäftstätigkeit und die Beziehung zu Geschäftspartnern haben kann, hervorgerufen durch interne oder externe Bedrohungen und Schwachstellen im Informationssystem. (s. KNOLL U. STRAHRINGER 2017, S. 4)

Innerhalb dieser Risikodiskussion spielen IT-Compliance-Risiken eine wichtige Rolle, die sich aus der Schnittmenge von IT-Risiken und Risiken durch Verstöße von externen Regelwerken oder rechtlichen Vorgaben ergeben. IT-Compliance befasst sich speziell mit den Risiken, die aus einer unzureichenden Funktion, Organisation, Nutzung, Verfügbarkeit, Sicherheit oder einem Missbrauch der IT resultieren können, wodurch gesetzliche Anforderungen oder unternehmensinterne Vorgaben potenziell verletzt werden. Die allgemeine Compliance geht darüber hinaus, indem sie potenzielle Regelverstöße als Risiken ansieht, die durch geeignete Maßnahmen im Rahmen des Risikomanagements behandelt werden müssen. Diese Auffassung von Compliance unterstreicht die Notwendigkeit, rechtliche Anforderungen nicht nur zu erfüllen, sondern auch proaktiv Risiken zu managen, die aus potenziellen Verstößen entstehen können. (s. KLOTZ U. DORN 2008)

Abbildung 2-24 zeigt den Zusammenhang von Risiken, IT-Risiken und IT-Compliance Risiken.

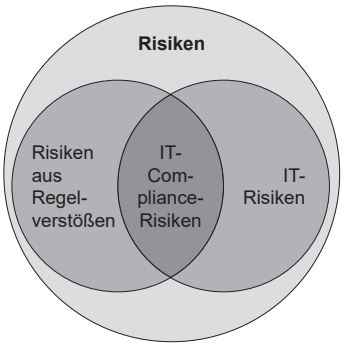

Abbildung 2-24: IT-Compliance-Risiken als Schnittmenge (s. KLOTZ U. DORN 2008)

Im Rahmen dieser Arbeit werden insbesondere die Risiken betrachtet, die mit dem Einsatz von Low-Code-Plattformen einhergehen (siehe Kapitel 6.1). Diese Risiken können sowohl technische IT-Risiken, Risiken aus Regelverstößen als auch Informationssicherheitsrisiken sein. Da das Informationssicherheitsrisiko oft in Zusammenhang mit Low-Code genannt wird, wird im folgenden Kapitel die Informationssicherheit spezifischer betrachtet.

2.3.5 Informationssicherheit

Das Hauptziel der Informationssicherheit besteht darin, Informationen unterschiedlichster Art und Herkunft vor unautorisiertem Zugriff oder Schaden zu schützen (s. HANSCHKE 2016, S. 205). Diese Informationen können auf verschiedenen Medien

gespeichert sein, sei es in physischer Form auf Papier, in elektronischen IT-Systemen oder auch im Gedächtnis der Benutzer*innen (s. BSI-STANDARD 200-1). Die IT-Sicherheit, als spezifischer Bereich der Informationssicherheit, fokussiert sich insbesondere auf den Schutz von elektronisch gespeicherten Informationen und den dazugehörigen Verarbeitungsprozessen (s. BSI-STANDARD 200-1).

Das Ziel von Informationssicherheit lässt sich anhand drei grundlegender Begriffe definieren:

- **Vertraulichkeit**: Informationen sollten nur für autorisierte Personen zugänglich sein und müssen vor unbefugtem Zugriff geschützt werden. Vertraulichkeit stellt sicher, dass sensible Daten nicht in die falschen Hände geraten.
- **Integrität**: Daten sollten genau und unverändert bleiben sollten. Maßnahmen zur Sicherstellung der Integrität gewährleisten den Schutz vor unbeabsichtigten oder böswilligen Änderungen.
- **Verfügbarkeit**: Informationen müssen für autorisierte Benutzer*innen verfügbar sein, wenn sie benötigt werden. Die entsprechenden Maßnahmen sorgen für einen kontinuierlichen Zugang zu Systemen und Daten, auch in Krisensituationen oder nach Zwischenfällen.

Diese drei Prinzipien bilden die Grundlage für die Entwicklung von Sicherheitsmaßnahmen und -kontrollen in einem Informationssicherheitsmanagementsystem (ISMS) (s. ISO 27001). Organisationen sind angehalten, ihre Praktiken der Informationssicherheit so zu gestalten, dass diese Grundprinzipien vollständig berücksichtigt werden, um ein adäquates Schutzniveau zu erreichen (s. ISO 27001). Die ISO 27002 dient dabei als Richtlinie für Informationssicherheitskontrollen und -praktiken und bietet Anleitungen sowie bewährte Verfahren für Informationssicherheitsmaßnahmen (siehe Kapitel 3.2.3). Indem Organisationen den Empfehlungen der ISO 27002 folgen, können sie eine detaillierte und praktische Umsetzung der durch die ISO 27001 vorgeschriebenen Sicherheitskontrollen realisieren.

SOMMERVILLE betrachtet Sicherheit aus organisatorischer Sicht auf den drei Ebenen Infrastruktursicherheit, Anwendungssicherheit und Betriebssicherheit.

- **Infrastruktursicherheit** befasst sich mit der Aufrechterhaltung der Sicherheit aller Systeme und Netzwerke, die eine Infrastruktur und einen Satz von gemeinsamen Diensten für die Organisation bereitstellen (s. SOMMERVILLE 2016, S.374-376).
- **Anwendungssicherheit** fokussiert sich auf die Sicherung einzelner Anwendungen oder Systemgruppen. Da Netzwerksysteme softwaregesteuert sind, können sie durch das Abfangen, Lesen oder Ändern von Netzwerkpaketen bedroht sein. Allerdings richten sich die meisten Sicherheitsangriffe auf Softwareinfrastrukturen, da Infrastrukturkomponenten wie Webbrowser allgemein zugänglich und auf Schwachstellen prüfbar sind.
- **Betriebssicherheit** konzentriert sich auf den sicheren Betrieb von Systemen und auf die Minimierung von Risiken, die bedingt durch menschliches

Fehlverhalten, die Sicherheit gefährden können. Dies umfasst das Bewusstsein für Sicherheitsrisiken und die Suche nach einem Gleichgewicht zwischen Sicherheit und Systemeffizienz.

Diese Arbeit konzentriert sich auf die Analyse der Informationssicherheit im Zusammenhang mit durch Low-Code-Plattformen entwickelten Anwendungen. Hierbei wird der Terminus „Informationssicherheit" spezifisch auf die IT-Sicherheit begrenzt und bezieht sich vorrangig auf den Schutz elektronisch gespeicherter Daten sowie die damit verbundenen Verarbeitungsprozesse innerhalb der mittels Low-Code erstellten Anwendungen. Bei der Integration der Low-Code-Plattform in die vorhandene IT-Infrastruktur ist die Infrastruktursicherheit technisch zu gewährleisten. Ebenso muss die Anwendungssicherheit bei der Entwicklung der Low-Code-Anwendung technisch sichergestellt werden. Organisatorische Aspekte der Betriebssicherheit sind dabei ganzheitlich zu berücksichtigen.

Dabei sind Ansätze problematisch, die IT-Sicherheit ausschließlich aus einer technischen Perspektive betrachten. In sozio-technischen Systemen, die aus sozialen (Menschen und Organisationen) und technischen Komponenten bestehen, ist Sicherheit ein zentrales Thema. Sicherheit kann nicht allein durch technische Mechanismen gewährleistet werden; sie erfordert eine Analyse aus sozialer und organisatorischer Perspektive (s. SALNITRI ET AL. 2014). Effektiv ist nur ein ganzheitlicher Ansatz, der IT-Sicherheit als Teil einer umfassenden IT-Governance begreift und die relevanten Compliance-Anforderungen berücksichtigt (s. RÜTER ET AL. 2010).

2.4 Abgrenzung des Untersuchungsbereichs

In einer wissenschaftlichen Arbeit ist es notwendig, bestimmte Aspekte der Realität zu vernachlässigen, um das Thema präzise zu fokussieren (s. MALONE U. CROWSTON 1994, S. 100). Diese Vorgehensweise entspricht dem systemtheoretischen Ansatz, auf dem diese Arbeit basiert (siehe Kapitel 1.4 und 4.2.1). Das Ziel dieses Kapitels ist es daher, auf Basis der in den vorherigen Abschnitten definierten Begriffe und Themen den Umfang der Untersuchung einzugrenzen und zu konkretisieren.

Das Ziel dieser Arbeit ist die Entwicklung eines Low-Code-Regelwerks, welches die bestehende IT-Governance erweitert. Diese Erweiterung ist notwendig, um Compliance-Anforderungen zu erfüllen und die Integrität sowie Sicherheit der IT-Systeme innerhalb der Organisation zu gewährleisten. Das zu entwickelnde Regelwerk zielt darauf ab, die mit Low-Code verbundenen Risiken zu minimieren. Diese Risiken können vielfältig sein und von Sicherheitslücken bis hin zu Integrationsproblemen mit bestehenden Systemen reichen. Durch die Implementierung zielgerichteter Richtlinien soll ein sicherer und effektiver Einsatz von Low-Code-Anwendung ermöglicht werden.

Es wird davon ausgegangen, dass Low-Code-Anwendung, abhängig von ihrem spezifischen Anwendungsfall, unterschiedliche Auswirkungen auf die IT-Architektur und die organisatorische Struktur haben können. Dies bedeutet, dass die Wahl der Low-Code-Plattform, die Art der Implementierung und die Integration in bestehende Systeme je nach Anwendungsfall variieren können. Aus den variierenden Auswirkungen von Low-

Code-Anwendung ergibt sich die Notwendigkeit, unterschiedliche Maßnahmen zur Erfüllung der im Regelwerk festgelegten Vorgaben zu ergreifen. Diese Maßnahmen müssen an den jeweiligen Anwendungsfall angepasst sein, um eine effektive Governance und Compliance sicherzustellen. Das Gestaltungsmodell, das in dieser Arbeit entwickelt wird, soll allgemeingültige sowie anwendungsspezifische Maßnahmen darstellen. Dieses Modell dient als Leitfaden für Unternehmen, um die Einführung von Low-Code-Anwendung strategisch zu planen und umzusetzen, indem es sowohl allgemeingültige als auch spezifische Maßnahmen für einzelne Anwendungstypen bereitstellt.

Tabelle 3 fasst diese Abgrenzung in einer morphologischen Darstellung zusammen, indem die für diese Arbeit relevanten Bereiche hervorgehoben werden.

Tabelle 3: Präzisierung des Untersuchungsbereichs (eigene Darstellung)

	Merkmal	**Ausprägungen**			
Anwendung	Anwendungsdomäne (s. FETTKE U. LOOS 2004)	Produzierende Unternehmen	IT-Dienstleister	Sonstige Dienstleistungen	Sonstiges
	Programmiersprache (s. WASZKOWSKI 2019)	1GL Maschinensprache	2GL Assemblersprache	3GL Programmierung	4GL Low-Code
	Integrationsperspektive (s. DERN 2011)	technisch	organisatorisch	wirtschaftlich	
	Unternehmensgröße (EU)	Mikrounternehmen	KMU	Großunternehmen	
Modellierung	Betrachtungsraum (s. WELTER 2006)	Merkmale	Typisierung		
	Modellierungszweck (s. ZELEWSKI 2008)	Beschreibung	Erklärung	Gestaltung	Metamodell
	Evaluation (s. FETTKE U. LOOS 2004)	sachlogisch	empirisch-qualitativ (Fallstudie)	empirisch-quantitativ (Studie)	
	Geltungsbereich (s. KNOBLICH 1969)	Anwendungsspezifisch	Generisch		

Legende	behandelt	teilweise behandelt	nicht behandelt

3 Stand der Erkenntnisse

Das zentrale Ziel der vorliegenden Arbeit wird in Kapitel 1.2 vorgestellt: Es soll ein Regelwerk entwickelt werden, um Low-Code-Anwendung in produzierenden Unternehmen nachhaltig in der IT-Architektur und der Organisationsstruktur zu verankern. Im Zuge der Vorbereitung dieser Arbeit erfolgte eine umfangreiche Literaturrecherche zum aktuellen Forschungsstand bezüglich des Einsatzes von Low-Code in produzierenden Unternehmen. Diese zeigt auf, dass kein Regelwerk für den Einsatz von Low-Code in Unternehmen existiert, welches umfangreich und gut anwendbar ist, jedoch einige Modelle sinnvoll verwendet werden können.

Um das beschriebene Ziel zu adressieren und den aktuellen Stand der Erkenntnisse darzulegen, gliedert sich das vorliegende Kapitel in zwei Hauptteile. Im ersten Teil werden existierende Studien zu Low-Code-Anwendungsfällen und Plattformen sowie deren Typisierung detailliert vorgestellt (siehe Kapitel 3.1). Im Anschluss daran konzentriert sich der zweite Teil auf die Analyse bestehender Arbeiten zur organisatorischen Verankerung von Low-Code-Anwendung innerhalb der Organisationsstruktur (siehe Kapitel 3.2).

3.1 Low-Code-Anwendungsfälle und Plattformen

Im folgenden Kapitel sollen über eine systematische Literaturrecherche Low-Code-Anwendungsfälle identifiziert und beschrieben werden. Ziel ist es, eine Liste von relevanten Low-Code-Anwendungsfällen zu erstellen, um auf dieser Grundlage eine Typisierung durchzuführen. Des Weiteren wird untersucht, ob und welche Ansätze für die IT-Governance im Kontext von Low-Code-Anwendung existieren (siehe Kapitel 3.2). Durch die Verbindung dieser beiden Untersuchungsbereiche werden sowohl die technischen Aspekte von Low-Code-Plattformen als auch die organisatorischen und strategischen Überlegungen, die für eine erfolgreiche Implementierung und Nutzung erforderlich sind, untersucht.

3.1.1 Low-Code-Anwendungsfälle

Für die systematische Literatursuche zur Identifizierung von Low-Code-Anwendungsfällen wurden die Datenbanken *Scopus, IEEE Xplore, Google Scholar* und *Web of Science* ausgewählt. Da Low-Code ein aktuelleres Thema ist, wurden nur wenige Suchbegriffe in der Literatursuche verwendet, um so viele relevante Quellen wie möglich zu identifizieren. Tabelle 4 veranschaulicht die gewählten Suchbefehle, um die Reproduzierbarkeit der Literaturrecherche sicherzustellen.

Tabelle 4: Suchbefehle der Literaturrecherche zu Low-Code-Anwendungsfällen (eigene Darstellung)

Digitale Datenbank	Suchbefehl
Scopus	(TITLE(low-code OR „low-code") AND TITLE-ABS-KEY (("use case" or "use cases") AND „manufacturing")
IEEE Xplore	(„Document Title":low-code OR „Document Title":"low code") AND ("All Metadata":"use case" OR "All Metadata":"use cases") AND („All Metadata": „manufacturing")
Google Scholar	Intitle:"low-code" OR intitle:"low code" AND ("use case" OR "use cases" OR Anwendungsfall OR Anwendungsfälle) AND (manufacturing OR „produzierende Unternehmen")
Web of Science	(TI=(low-code OR „low-code")) AND ALL=("use case" OR "use cases" OR "Anwendungsfall" OR "Anwendungsfälle") AND (manufacturing OR „produzierende Unternehmen")

Durch den Einsatz gezielter Suchstrategien konnten insgesamt 47 Publikationen identifiziert werden, die den Anforderungen der Literaturrecherche entsprechen. Um die Menge der relevanten Arbeiten weiter zu präzisieren und die Qualität der Untersuchung zu sichern, wurden zusätzliche Qualitätssicherungskriterien angewendet. Diese Kriterien wurden spezifisch ausgewählt, um sicherzustellen, dass die in die Untersuchung einbezogenen Beiträge einen direkten Beitrag zur Beantwortung der formulierten Forschungsfrage leisten. Die angewandten Qualitätssicherungskriterien sind:

- **Qualitätssicherungskriterium Q1**: Dieses Kriterium gewährleistet, dass nur Veröffentlichungen berücksichtigt werden, die im Jahr 2019 oder später erschienen sind. Angesichts der raschen Entwicklung und des hohen Innovationsgrades in der Low-Code-Plattform wird ein Untersuchungszeitraum von fünf Jahren als ausreichend angesehen, um relevante und aktuelle Erkenntnisse für die Dissertation zu gewinnen.
- **Qualitätssicherungskriterium Q2**: Mit diesem Kriterium werden gezielt Beiträge erfasst, die sich mit Low-Code-Anwendungsfällen in produzierenden Unternehmen befassen. Diese thematische Eingrenzung ist erforderlich, um Publikationen auszuschließen, die trotz der Verwendung spezifischer Suchbegriffe nicht unmittelbar zum gewünschten Themengebiet beitragen.

Abbildung 3-1 gibt einen Überblick über das Vorgehen der systematischen Literatursuche.

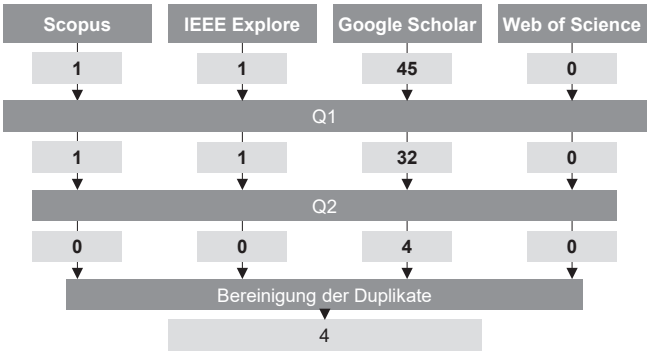

Abbildung 3-1: Vorgehen zur Literaturrecherche zu Low-Code-Anwendungsfällen (eigene Darstellung)

Der zweite Schritt der Literaturrecherche war die Bewertung der identifizierten Quellen. Ziel dieser Bewertung war es, die Relevanz der Ergebnisse in Bezug auf die vorliegende Arbeit zu untersuchen. Zu diesem Zweck wurde geprüft, ob die Low-Code-Programmierung im Fokus der Ergebnisse steht, produzierende Unternehmen Gegenstand der Untersuchung sind, vorgestellte Anwendungsfälle spezifisch erläutert werden und ob Ansätze zur Typisierung von Low-Code-Anwendungsfällen vorgestellt werden. Dazu werden die Quellen im Folgenden zunächst vorgestellt und anschließend die Erkenntnisse zusammengefasst.

„A Low-Code Development Platform for constructing industrial apps" (WANG ET AL. 2021):

WANG ET AL. präsentieren ein Framework zur Erstellung von Industrieapplikationen mit Low-Code, das die Nutzungsschwelle und Wartungskosten reduzieren soll. Sie validieren das Framework mit einer Applikation zur vorausschauenden Wartung von Flugzeugturbinen, wodurch die Effektivität des Frameworks demonstriert wird. Abbildung 3-2 zeigt die Systemarchitektur des „Low-Code Development Frameworks".

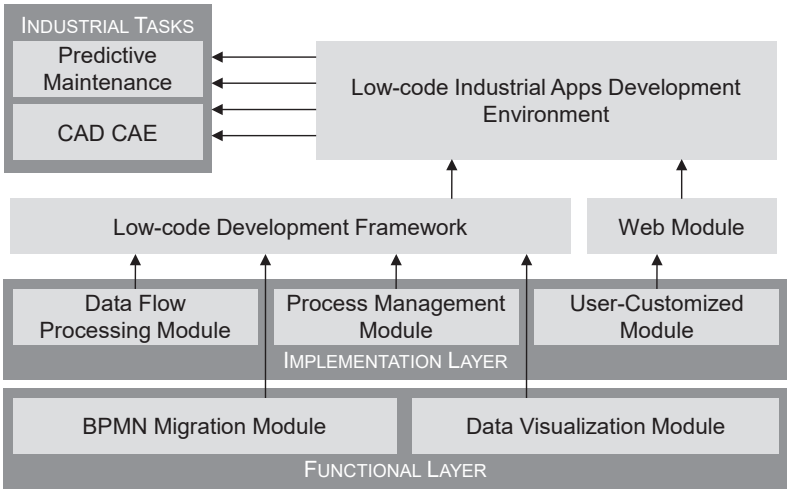

Abbildung 3-2: Systemarchitektur des "Low-Code Development Frameworks" (s. WANG ET AL. 2021)

Das vorgeschlagene Framework zielt darauf ab, die Produktivität bei der Entwicklung industrieller Anwendungen zu steigern, indem es eine offene, einheitliche Low-Code-Entwicklungsumgebung bereitstellt. Es integriert allgemeines industrielles Wissen und grundlegende Komponenten für die Entwicklung industrieller Anwendungen. Die Studie demonstriert die Funktionalität des Frameworks anhand eines Anwendungsfalls zur prädiktiven Wartung, unter Verwendung von Daten über die Lebensdauer von Flugzeugturbinen zur Überprüfung der Wirksamkeit des Systems. Die Hauptbeiträge des Papers umfassen die Analyse und Zusammenfassung von Gemeinsamkeiten typischer industrieller Aufgaben und die Entwicklung einer Reihe von Modulen und Komponenten mit genereller Anwendbarkeit. Darüber hinaus wird der Einsatzfall einer prädiktiven Wartung als typisches industrielles Feld genutzt, um die Effektivität des Frameworks zu überprüfen. Zusammenfassend stellt diese Arbeit einen bedeutenden Schritt in Richtung der Vereinfachung der Entwicklung industrieller Anwendungen dar.

Allerdings liegt der Fokus des Papers auf der Entwicklung von Anwendungsfällen, nicht auf den Anwendungsfällen selbst. Der Anwendungsfall wird nicht detailliert genug beschrieben, um beschreibende Merkmale für Anwendungsfälle abzuleiten, die allgemeingültig sind. Außerdem fehlt es an Ansätzen zur Typisierung von Low-Code-Anwendungsfällen. Insgesamt unterstützt das Framework dabei, die Integration von Low-Code-Plattformen in die IT-Architektur nachzuvollziehen. Diese Aspekte können verwendet werden. Detailliertere Erkenntnisse zu der Strukturierung von Low-Code-Anwendungsfällen behandelt das Paper nicht.

„Low-Code-Programmierung als Ansatz zur Gestaltung bedarfsgerechter informatorischer Assistenzsysteme – eine Fallstudie." (ADRIAN ET AL. 2020b)

ADRIAN ET AL. beschreiben die Entwicklung eines Montageassistenzsystems mittels einer Low-Code-Plattform. Dieses Assistenzsystem zielt darauf ab, die Komplexität in der Montage durch Bereitstellung digitaler Informationen zu reduzieren, Montagefehler zu verringern und den administrativen Aufwand zu minimieren. Als Ergebnis des Projekts konnte eine prototypische Softwarelösung mittels einer Low-Code-Plattform effizient entwickelt werden. Es wird gezeigt, dass Low-Code-Programmierung eine praxistaugliche Methode darstellt, um bei der Erstellung von Anwendungen die Entwicklungsgeschwindigkeit zu erhöhen und gleichzeitig den Programmieraufwand zu verringern.

Ein wichtiger Aspekt der Studie ist die systematische Darstellung des Entwicklungsprozesses der Software, von der Ermittlung der Anforderungen über die Erstellung des Informationsmodells bis hin zur Umsetzung der Softwarelösung. Abschließend gibt die Studie wertvolle Hinweise für die betriebliche Praxis und betont die Vorteile der Low-Code-Programmierung: Die Möglichkeit, Software schnell und einfach zu gestalten, Wartezeiten auf IT-Dienstleistungen zu vermeiden und passgenaue Lösungen zu finden. Gleichzeitig werden die Grenzen der Low-Code-Programmierung aufgezeigt, etwa in Bezug auf die begrenzten Programmierfunktionalitäten und grafischen Darstellungsmöglichkeiten sowie Herausforderungen bei der Schnittstellengestaltung zu ERP-Systemen.

Insgesamt bietet die Studie einen umfassenden Einblick in die Potenziale und Herausforderungen der Low-Code-Programmierung für die Entwicklung von betriebsspezifischen Softwareanwendungen. Dieser Anwendungsfall hat große Relevanz für die Produktion und wird spezifisch genug beschreiben. Insbesondere die Beschreibung über mögliche Schnittstellen sowie die Grenzen der Low-Code-Programmierung finden sich in dieser Arbeit in Form von Merkmalen wieder (siehe Kapitel 5.2.2). Allerdings lässt sich auch hier kein Ansatz zur Typisierung finden.

„App Development via Low-Code-Programming as Part of Modern Industrial Engineering Education" (ADRIAN ET AL. 2020a)

ADRIAN ET AL. analysieren in dieser Untersuchung die Integration von Low-Code-Programmierung in die Ausbildung von Wirtschaftsingenieuren. Ein Anwendungsfall, bei dem Studenten eine Applikation zur Qualitätssicherung von Spielzeugfahrzeugen programmieren, wird betrachtet. Obwohl dieser Anwendungsfall in der Produktion von Bedeutung ist, kann der Anwendungsfall nicht weiter analysiert werden, da er nicht repräsentativ für den realen Prozess der Qualitätskontrolle ist und keine Typisierungsansätze vorhanden sind.

„Generating customized low-code development platforms for digital twins" (DALIBOR ET AL. 2022)

DALIBOR ET AL. beschreiben die Erstellung einer Low-Code-Plattform zur Entwicklung digitaler Zwillinge, die als hochpräzise digitale Modelle reale Objekte abbilden. Die Grundlage dieses Ansatzes kombiniert drei Schlüsselkomponenten:

- **Eine Code-Generierungsinfrastruktur** für Informationssysteme, die die schnelle Entwicklung von Plattformen für digitale Zwillinge ermöglicht.
- **Eine erweiterbare Basisarchitektur** für selbstadaptive digitale Zwillinge, die sicherstellt, dass sich die digitalen Zwillinge basierend auf Echtzeitdaten und sich ändernden Bedingungen anpassen können.
- **Wiederverwendbare Sprachkomponenten** für die Konfiguration von digitalen Zwillingen, um die Anpassung und Erweiterung der Fähigkeiten digitaler Zwillinge gemäß spezifischen Domänenanforderungen zu erleichtern.

Der Prozess beinhaltet zunächst, dass Softwareingenieur*innen das Informationssystem mit den notwendigen Modellierungssprachen konfigurieren, um die Low-Code-Plattform zu generieren. Anschließend nutzen Fachexpert*innen die generierte Plattform, um digitale Zwillinge zu erstellen, wobei sie die maßgeschneiderten Fähigkeiten der Plattform für ihre spezifischen Anwendungen nutzen. Dieser zweistufige Ansatz vereinfacht die Erstellung und den Betrieb von angepassten digitalen Zwillingen in verschiedenen Bereichen.

Das Paper beschreibt die Validierung anhand eines Anwendungsfalls im Zusammenhang mit dem Spritzgießen. Durch den Anwendungsfall wird demonstriert, wie die Low-Code-Plattform von Fachexpert*innen genutzt werden kann, um einen digitalen Zwilling für das Spritzgießen zu modellieren. Insgesamt bietet dieses Paper eine strukturierte Methode für die Nutzung von Low-Code-Plattformen in der Entwicklung und Verwaltung von digitalen Zwillingen an und betont die Bedeutung des modellgetriebenen Engineerings beim Vereinfachen komplexer Softwareentwicklungsaufgaben für Fachexpert*innen. Die Erkenntnisse aus der Modellierung werden bei der Bildung von Merkmalen für Anwendungsfalltypen verwendet. Jedoch wird der Anwendungsfall nicht spezifisch genug beschrieben, um Merkmale für eine Typisierung abzuleiten.

Innerhalb einer Literaturrecherche konnte identifiziert werden, dass Low-Code-Anwendung in produzierenden Unternehmen verwendet werden. Es ist jedoch bemerkenswert, dass die bestehende Literatur oft keine umfassenden Beschreibungen spezifischer Anwendungsfälle enthält und derzeit keine etablierte Typisierung für Low-Code-Anwendungsfälle in produzierenden Unternehmen existiert. Daher wird im Folgenden der Blickwinkel erweitert und es wird untersucht, inwiefern Typisierungen von Low-Code-Plattformen bereits untersucht wurden.

3.1.2 Typisierung von Low-Code-Plattformen

In diesem Kapitel soll analog zu der in Kapitel 3.1.1 vorgestellten Methodik eine systematische Literaturrecherche erfolgen, um nun *Ansätze zur Typisierung* von Low-

Code-Plattformen zu identifizieren. Hierfür wurden die Datenbanken *Scopus, IEEE Xplore, Google Scholar* und *Science Direct* genutzt.

Tabelle 5 veranschaulicht die gewählten Suchbefehle, um die Reproduzierbarkeit der Literaturrecherche sicherzustellen.

Tabelle 5: Suchbefehle der Literaturrecherche zur Typisierung von Low-Code-Plattformen (eigene Darstellung)

Digitale Datenbank	Suchbefehl
Scopus	TITLE-ABS-KEY (low-code) OR TITLE-ABS-KEY (low-code/no-code) AND TITLE-ABS-KEY (typification OR typology OR classification)
IEEE Xplore	("Abstract": low-code OR "Abstract": low-code/no-code OR "Document Title": low-code OR "Document Title": low-code/no-code) AND ("All Metadata": typification OR "All Metadata": typology OR "All Metadata": classification)
Google Scholar	intitle:("low-code" OR "low-code/no-code") AND („typification“ OR “typology“ OR "taxonomy" OR „classification“ OR “Typisierung" OR “Klassifizierung“)
Science Direct	TITLE-ABS-KEY ("low-code" OR "low-code/no-code") AND TITLE ("low-code" OR "low-code/no-code") AND TERMS (typification OR typology OR classification)

Durch den Einsatz der Suchstrategien konnten insgesamt 237 Publikationen identifiziert werden. Zur weiteren Eingrenzung wurden die Qualitätssicherungskriterien analog zu der vorherigen Recherche in Kapitel 3.1.1 gesetzt.

- **Qualitätssicherungskriterium Q1**: Dieses Kriterium gewährleistet, dass nur Veröffentlichungen berücksichtigt werden, die im Jahr 2019 oder später erschienen sind. Angesichts der raschen Entwicklung und des hohen Innovationsgrades in der Low-Code-Plattform wird ein Untersuchungszeitraum von fünf Jahren als ausreichend angesehen, um relevante und aktuelle Erkenntnisse für die Dissertation zu gewinnen.
- **Qualitätssicherungskriterium Q2**: Mit diesem Kriterium werden gezielt Beiträge erfasst, die sich mit Low-Code-Plattformen für produzierenden Unternehmen befassen. Diese thematische Eingrenzung ist erforderlich, um Publikationen auszuschließen, die trotz der Verwendung spezifischer Suchbegriffe nicht unmittelbar zum gewünschten Themengebiet beitragen.

Abbildung 3-3 zeigt das Vorgehen sowie die Ergebnisse der Literaturrecherche

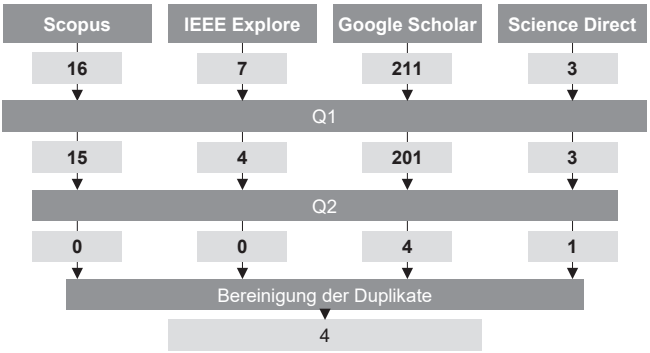

Abbildung 3-3: Vorgehen der Literaturrecherche zur Typisierung von Low-Code-Plattformen (eigene Darstellung)

In einem nächsten Schritt wurde untersucht, inwiefern diese Ansätze für die Typisierung von Low-Code-Plattformen für den vorliegenden Forschungsrahmen dienen.

„A taxonomy of industrial IoT platforms" architectural features" (ARNOLD ET AL. 2021)

ARNOLD ET AL. entwickelten eine Typisierung der architektonischen Funktionalitäten von „Industrial Internet-of-Things" (IIoT) Plattformen. Grundlage ihrer Typologie bildeten Expert*inneninterviews und die Analyse von 50 IIoT Plattformen, dargestellt in einem morphologischen Kasten in Abbildung 3-4.

	Dimension		Characteristics			
Infrastructure Layer	Hardware Support	ME	Certified Hardware		Hardware-Agnostic	
	Platform Hosting	NE	On-Premise	Public Cloud	Private Cloud	Hybrid
	Data Processing	NE	Edge	Fog	On-Platform	
Network Layer	Physical Data Transportation	NE	Wired	Short-Range Wireless	Cellular	LPWAN
	Logical Data Transmission	NE	Internet Protocols	IoT-Specific Protocols	Industry-Specific Protocols	
Middleware Layer	Data Structure	NE	Structured		Unstructured	
	Analytics Types	NE	Descriptive	Real-Time	Predictive	Prescriptive
	Analytics Technology	ME	Basic		Advanced	
	External Integration	NE	Business	Machine	Web Services	
	Platform Source Code	ME	Open Source	Open Components	Closed Source	
Application Layer	APIs	ME	Standardized APIs		Custom APIs	
	Application Deployment	NE	Platform-Native	Containerized	Off-Platform	
	Marketplace	NE	Internal Marketplace	External Marketplace	No Marketplace	

Abbildung 3-4: Taxonomie der Architekturmerkmale von IIoT-Plattformen (s. ARNOLD ET AL. 2021)

Die Autor*innen leiteten daraus vier Archetypen von IIoT Plattformen ab: Allrounder, Purist, Analyst und Verbinder. Diese Archetypen dienen zur Vereinheitlichung typischer IIoT-Plattformen und werden in dieser Arbeit als Synonym für den Begriff der Typen verwendet.

In Abbildung 3-4 wird deutlich, dass die Eigenschaften und Merkmale systematisch entlang von Systemebenen organisiert wurden, die in der äußersten linken Spalte des morphologischen Kastens aufgelistet sind. Diese Strukturierung ermöglicht es, einen klaren Überblick über die spezifischen Aspekte und Funktionalitäten verschiedener Ebenen der Systemarchitektur zu erhalten. Obwohl die meisten Merkmale von IIoT-Plattformen (Industrial Internet of Things) nicht auf Low-Code-Anwendungsfälle anwendbar sind, liefert die „Application Layer“ eine Grundlage für potenzielle Merkmale von Low-Code-Anwendungsfällen. Insbesondere das Merkmal und die Ausprägungen zu “APIs” finden sich in der Definition der Anwendungsfälle unter dem Merkmal „Schnittstellen“ wieder (siehe Kapitel 5.2.2).

„Drivers and Evolution Paths of BPMS: State-of-the-Art and Future Research Directions” (SZELĄGOWSKI ET AL. 2022)

SZELĄGOWSKI ET AL. analysierten die treibenden Kräfte und Hindernisse in der Entwicklung von Business Process Management Systemen (BPMS) und stellten praxisnahe Empfehlungen sowohl für Anbieter als auch Nutzer von BPMS-Software zur Verfügung. Für Anbieter empfehlen sie, den Fokus auf technologische Offenheit, Zugänglichkeit für Nutzer (mittels Low-Code-Plattformen) sowie auf die Integration von Komponenten für Wissensmanagement zu legen, um die Effektivität von BPMS zu steigern. Nutzern wird geraten, eine umfassende Analyse ihrer Geschäftsprozesse vorzunehmen und BPM-Werkzeuge sorgfältig auszuwählen, die zur Prozessbeschaffenheit und den Kompetenzen der Belegschaft passen. Diese Vorschläge zielen darauf ab, BPM-Implementierungen als ganzheitliche Unterfangen zu gestalten, die sowohl technische Möglichkeiten als auch den organisatorischen Rahmen berücksichtigen.

In der durchgeführten Analyse wurden Klassifizierungen von Plattformen dargelegt, die die Entwicklung von BPMS (Business Process Management Systems) Softwareanwendungen mittels Low-Code-Plattform ermöglichen. Die technischen Merkmale sind allerdings zu BPMS spezifisch, um eine sinnvolle Anwendung für das Erarbeiten von Low-Code-Merkmalen zu finden. Dennoch lassen sich die praxisnahen Empfehlungen für die Nutzenden von BPMS-Software anwenden. Die Berücksichtigung der Kompetenzen der Mitarbeitenden für die Auswahl der BPMS-Werkzeuge findet sich als Berücksichtigung der Programmierfähigkeiten in Kapitel 5.2.3 wieder.

Zukünftig fordert das Paper Methoden, die die Implementierung von BPM enger an organisatorische Strategien knüpfen. Es wird empfohlen, die Forschung im Bereich der organisatorischen und technischen Integration von Unternehmenssoftware weiter voranzutreiben. Damit unterstreicht das Paper einen Forschungsbedarf, der im Rahmen dieser Arbeit aufgegriffen und adressiert wird.

„Low-code engineering for internet of things: a state of research" (IHIRWE ET AL. 2021)

IHIRWE ET AL. untersuchten die Verwendung von Low-Code-Plattformen für die Entwicklung von Softwareanwendungen für IoT-Systeme. Das Paper präsentiert eine Analyse von sechzehn Plattformen, um die Funktionalitäten zu verstehen, die von bestehenden Low-Code-Plattformen für die IoT-Entwicklung unterstützt werden. Die Funktionalitäten und deren Ausprägungen sind in Tabelle 6 aufgeführt.

Tabelle 6: Low-Code-Funktionalitäten für die IoT-Entwicklung (eigene Darstellung i. A. a. IHIRWE ET AL. 2020)

Funktionalität	Ausprägung			
Unterstützung der Anforderungsmodellierung	Anforderungsspezifikation		Anforderungsverifikation	
Unterstützung der Domänenmodellierung	Grafische Benutzeroberfläche	Strukturmodell	Verhaltensmodell	Multi-View-Modellierung
Unterstützung von Test und Verifikation	Testumgebung	Modellprüfung	Modellvalidierung	
Analyseumgebung	Echtzeitanalyse	Zuverlässigkeit	System-QoS	
Wiederverwendbarkeit	Export von Artefakten	Import von externen Artefakten	Artefaktspeicherung und zukünftige Wiederverwendung	
Bereitstellungs-unterstützung	Lokale Bereitstellung	Cloud-Bereitstellung	Code-Generierung	Laufzeitanpassung
Interoperabilität	API-Integration		Verbindung zu externen Datenquellen	
Erweiterbarkeit	Einfaches Hinzufügen von Modellierungsfunktionen		Einfaches Modifizieren bestehender Funktionen	
Zusätzliche Merkmale	Fokus auf die "Thing"-Ebene	Fokus auf die Anwendungsebene	Open Source für Entwickler	
Zielunterstützung	Unterliegende Infrastruktur	Zielplattform	Code-Generierungssprache	

Der Fokus der entwickelten Typologie liegt auf Funktionalitäten wie Unterstützung der Domänenmodellierung, Unterstützung von Test- und Verifikation, Bereitstellungsunterstützung, Interoperabilität und Erweiterbarkeit. Darüber hinaus werden die Grenzen aktueller Ansätze diskutiert und mögliche Wege, um diese Einschränkungen in Zukunft zu verbessern und anzugehen, vorgestellt.

Trotz der Bezeichnung ihrer Ergebnisse als Typologie durch die Autor*innen stellt die Studie für die spezifische Forschungsfrage dieser Arbeit bezüglich einer Typisierung von Low-Code-Anwendungsfällen keinen direkten Mehrwert dar. Der Grund dafür liegt insbesondere darin, dass sich die Untersuchung primär auf Anbieter von Low-Code-Plattformen konzentriert und weniger auf die damit zu entwickelnden Anwendungsfälle.

„Supporting the understanding and comparison of low-code development platforms" (SAHAY ET AL. 2020)

SAHAY ET AL. klassifizieren in ihrer Arbeit eine Reihe von bekannten Low-Code-Plattformanbietern, darunter OutSystem, Mendix und Microsoft Power Apps, nach ihren technischen Funktionalitäten. Eine Unterscheidung der Funktionalitäten erfolgt zunächst durch die Ebenen der Systemarchitektur, der Applikationsebene, der Dienstleistungs-Integrationsebene, der Daten-Integrationsebene und der Bereitstellungsebene. Daraufhin erläutern die Forschenden die Hauptkomponenten einer Low-Code-Entwicklungsplattform und die typischen Prozessschritte bei der Low-Code-Softwareentwicklung (siehe Kapitel 2.1.3). Anhand eines Merkmaldiagramms, das die Hauptabweichungen bei den Funktionalitäten von Low-Code-Entwicklungsplattformen beschreibt, werden die Funktionalitäten der Plattformanbieter unterschieden. Hierbei bestimmen SAHAY ET AL. konkret, ob die Plattform eines Anbieters eine bestimme Funktionalitäts-Ausprägung aufweist oder nicht. Tabelle 7 zeigt die untersuchten Funktionalitätsausprägungen.

Tabelle 7: Funktionalitätsausprägungen von Low-Code-Plattformen (eigene Darstellung i. A. a. SAHAY ET AL. 2020)

Funktionalität	Ausprägung				
Grafische Benutzeroberfläche	Drag-and-drop Designer	Point-and-click Ansatz	Vorabgefertigte Formulare/Berichte	Vorabgefertigte Dashboards	Formulare
	Fortschrittsverfolgung	Erweiterte Berichterstellung	Eingebaute Workflows	Konfigurierbare Workflows	
Interoperabilität Unterstützung	Interoperabilität mit externen Diensten		Verbindung mit Datenquellen		
Sicherheitsunterstützung	Anwendungssicherheit		Plattformsicherheit		
Unterstützung für kollaborative Entwicklung	Offline-Zusammenarbeit		Online-Zusammenarbeit		
Wiederverwendbarkeit Unterstützung	Eingebaute Workflows	Vorabgefertigte Formulare/Berichte	Vorabgefertigte Dashboards		
Skalierbarkeit	Skalierbarkeit nach Anzahl der Benutzer	Skalierbarkeit des Datenverkehrs	Skalierbarkeit des Datenspeichers		
Geschäftslogik-Spezifikationsmechanismen	Business Rules Engine	Grafischer Workflow-Editor	KI-gestützte Geschäftslogik		
Mechanismen zur Anwendungserstellung	Code-Generierung		Modelle zur Laufzeit		
Bereitstellungsunterstützung	Bereitstellung in der Cloud		Bereitstellung auf lokalen Infrastrukturen		
Arten unterstützter Anwendungen	Ereignisüberwachung	Prozessautomatisierung	Genehmigungsprozesskontrolle	Eskalationsmanagement	
	Bestandsmanagement	Qualitätsmanagement	Workflow-Management		

Das Paper liefert zwar umfassende Einblicke in die technischen Funktionalitäten von Low-Code-Plattformen, doch sind die Ergebnisse für die Zwecke dieser Arbeit zu stark auf technische Details fokussiert. Auch wenn die Funktionalitäten zu detailliert sind, um sie für die folgende Arbeit zu verwenden, dienen diese sowie der beschriebene Low-Code-Entwicklungsprozess als Anhaltspunkt, um bestimmte Merkmale bei der Implementierung und dem Betrieb von Low-Code-Anwendung zu berücksichtigen. Das

Merkmal „Skalierbarkeit bezüglich der Nutzeranzahl“ findet sich im Merkmal „Nutzung“ wieder (vgl. Kapitel 5.2.1).

3.1.3 Zusammenfassung

Zusammenfassend lässt sich festhalten, dass aktuelle Forschungen und Literatur im Bereich Low-Code-Entwicklung hauptsächlich technische Aspekte der Low-Code-Programmierung und deren Anwendung in Produktionsunternehmen untersuchen. Spezifische Anwendungsfälle werden teilweise berücksichtigt, jedoch gibt es eine Lücke in der Forschung, wenn es um die Typisierung von Low-Code-Plattformen und -Anwendungsfällen geht, die sowohl technische als auch organisatorische Merkmale umfassen. Insbesondere die Entwicklung eines Ansatzes zur Typisierung von Low-Code-Plattformen und -Anwendungsfällen ist bisher nicht Gegenstand der Untersuchungen.

Daraus ergibt sich die Notwendigkeit, Forschungsansätze zur Low-Code-Governance zu identifizieren, um zusätzliche organisatorische Aspekte von Low-Code aus der wissenschaftlichen Forschung zu extrahieren. Diese Erkenntnisse sollen genutzt werden, um die Forschungslücke zu schließen und ein umfassenderes Verständnis von Low-Code-Anwendungsfällen und der Implementierung und dem Betrieb dieser zu entwickeln, das sowohl technische als auch organisatorische Dimensionen berücksichtigt.

3.2 Low-Code-Governance

Im folgenden Kapitel werden zunächst relevante Forschungsarbeiten zum Thema Low-Code-Governance dargestellt und in den Kontext dieser Arbeit eingeordnet (siehe Kapitel 3.2.1). Im darauffolgenden Abschnitt werden zusätzliche Modelle erörtert, die Hinweise für die Merkmale und Typisierungen von Low-Code-Anwendungsfällen geben und somit einen Beitrag zum ersten Beschreibungsmodell leisten. Abschließend werden in Kapitel 3.2.3 IT-Governance-Modelle analysiert, die Orientierungshilfen für die Strukturierung von Low-Code-Regeln bieten können und im zweiten Beschreibungsmodell in Form eines strukturierten Regelwerks zur Anwendung kommen.

3.2.1 Konzepte von Low-Code-Governance

Die Datenerhebung für Konzepte von Low-Code-Governance erfolgte durch die Nutzung verschiedener Datenbanken. Dabei wurden die bibliographischen Online-Bibliotheken *Scopus, IEEE Xplore* und *Springer Link* sowie die Suchmaschine *Google Scholar* verwendet. Für die Literaturrecherche wurden die Begriffe „Low-Code“, „Governance“, „Regeln“ und „Regelwerk“ verwendet. Diese Methodik und die verwendeten Suchbefehle dienen der Sicherstellung der Reproduzierbarkeit der Literaturrecherche (siehe Tabelle 8).

Tabelle 8: Suchbefehl der Literaturrecherche zu Low-Code-Governance (eigene Darstellung)

Digitale Datenbank	Suchbefehl
SCOPUS	TITLE-ABS-KEY (low-code) AND TITLE-ABS-KEY (governance OR rules OR ruleset)
IEEE Xplore	(„Abstract“: low-code OR „Abstract“:Low-code OR ("Abstract": low code) AND („Abstract“:governance OR „Abstract“:ruleset OR „Abstract“: rules))
Google Scholar	allintitle: governance OR rule OR ruleset „low-code“
Springer Link	Where the title contains (low-code and ("governance" or "rules" or "ruleset"))

Die Recherche identifizierte 49 Arbeiten, von denen nach analogen Qualitätssicherungskriterien wie in Kapitel 3.1.1 und 3.1.2 ausgewählt wurde:

- **Qualitätssicherungskriterium Q1**: Dieses Kriterium gewährleistet, dass nur Veröffentlichungen berücksichtigt werden, die im Jahr 2019 oder später erschienen sind. Angesichts der raschen Entwicklung und des hohen Innovationsgrades in der Low-Code-Plattform wird ein Untersuchungszeitraum von fünf Jahren als ausreichend angesehen, um relevante und aktuelle Erkenntnisse für die Dissertation zu gewinnen.
- **Qualitätssicherungskriterium Q2**: Mit diesem Kriterium werden gezielt Beiträge erfasst, die sich mit Low-Code und Governance-Ansätzen befassen. Diese thematische Eingrenzung ist erforderlich, um Publikationen auszuschließen, die trotz der Verwendung spezifischer Suchbegriffe nicht unmittelbar zum gewünschten Themengebiet beitragen.

Duplikate wurden im abschließenden Schritt entfernt, was zu vier relevanten Publikationen führte. Diese liefern wichtige Erkenntnisse für die Dissertation. Die ausgewählte Literatur konzentriert sich hauptsächlich auf wissenschaftliche Arbeiten und ausgewählte praxisorientierte Ansätze, wobei betriebliche Beiträge als weniger relevant angesehen werden (vgl. GÖTZEN 2022). Die Datenerhebung ist in Abbildung 3-5 dargestellt.

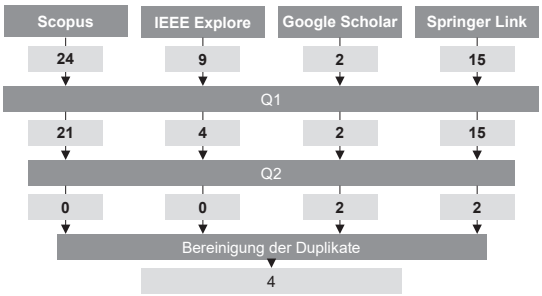

Abbildung 3-5: Vorgehen zur Literaturrecherche zu Low-Code-Governance (eigene Darstellung)

Im Folgenden werden die identifizierten Quellen detaillierter beschrieben.

"Democratizing Software Development: A Systematic Multivocal Literature Review and Research Agenda on Citizen Development" (BINZER U. WINKLER 2022)

Das Paper untersucht das Konzept des Citizen Developers im Kontext der Softwareentwicklung. Es wird eine systematische, multivokale Literaturübersicht durchgeführt, um das Phänomen des Citizen Development zu erforschen. Die Studie identifiziert sechs Schlüsselthemen (siehe Abbildung 3-6):

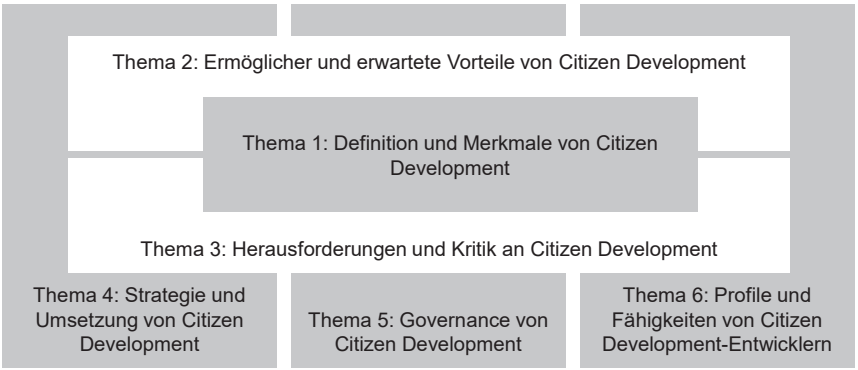

Abbildung 3-6: Konzeptuelles Modell der aktuellen Literaturthemen zu Citizen Development (s. BINZER U. WINKLER 2022)

In Anbetracht der Forschungsziele dieser Arbeit wird im Folgenden insbesondere auf die Herausforderungen und Kritik, die Strategie und Implementierung, die Governance sowie die Profile und Fähigkeiten von Citizen Developern eingegangen.

Herausforderungen und Kritik

Die identifizierten Herausforderungen umfassen sowohl die Grenzen von Low-Code-Plattformen als auch die Kompetenzen der Citizen Developer. Das Paper hebt hervor, dass die Softwareentwicklung mittels Low-Code-Plattformen von vielen Unternehmen weiterhin als anspruchsvolle Aufgabe angesehen wird, die spezialisierte Kenntnisse benötigt. Zusätzlich werden Integrationsprobleme und Bedenken hinsichtlich der Qualität wegen mangelhafter Testverfahren als signifikante Hindernisse betrachtet. Sicherheitsrisiken sowie das fehlende Wissen der Citizen Developer über Governance-Praktiken oder etablierte Standards werden ebenfalls als kritische Punkte beleuchtet.

Strategie und Implementierung

Die Strategie und Implementierung diskutiert, wie Organisationen Citizen Development in der Praxis umsetzen können. In diesem Zusammenhang wurden Faktoren untersucht, die die Entscheidung eines Unternehmens zur Annahme von Citizen Development bestimmen. Aktive Unterstützung des Top-Managements, zentralisierte IT-Governance und die Einbeziehung externer Partner wurden als positiv für die Annahme von Citizen Development erkannt.

Governance

Bei der Governance wurde die Zusammenarbeit zwischen Geschäfts- und IT-Abteilungen untersucht. Viele Artikel befürworten die Integration von Citizen Development-Programmen in zentralisierte IT-Governance-Strukturen, um Sicherheit, Compliance, Datenintegrität und Effizienz zu gewährleisten.

Profile und Fähigkeiten

Das sechste und letzte Thema befasst sich mit den Profilen und Fähigkeiten individueller Citizen Developer. Aufgrund ihrer tiefen und unmittelbaren Geschäftsexpertise wird Citizen Developern ein gründliches Verständnis der täglichen Geschäftsanforderungen und bestehender operativer Ineffizienzen zugeschrieben. In diesem Kontext werden Problemlösungsfähigkeiten oft als Schlüsselkompetenz von Citizen Developern angesehen.

Das Paper schlägt die weitere Erforschung effektiver Governance-Modelle, der Auswirkungen des Citizen Developments auf die Organisationskultur und -prozesse sowie die Entwicklung von Best Practices zur Unterstützung von Citizen Developern vor. Die Erkenntnisse aus dem Paper bieten sowohl für die Typisierung von Anwendungsfällen als auch für die Definition von Regeln Ansatzpunkte. Bei der Typisierung von Anwendungsfällen werden vor allem die Empfehlungen zur Strukturierung hinsichtlich der Komplexität der Anwendungsentwicklung, der Kooperation zwischen Geschäfts- und IT-Bereichen sowie der Kompetenzen von Citizen Developern berücksichtigt. Die im Paper herausgestellten Herausforderungen im Umgang mit Low-Code-Plattformen und die Leitlinien zur Low-Code-Governance leisten einen wichtigen Beitrag zur Ableitung entsprechender Regeln.

„Approach to the Implementation of Low-Code Platforms" (ROGOZOV ET AL. 2023)

In der Studie wird die Zusammenarbeit zwischen der Geschäftsseite und der IT im Kontext von Low-Code-Plattformen thematisiert. Der Nutzer repräsentiert das System im Konzept der Frage wie Geschäftsprobleme mithilfe des Systems gelöst werden. Daher muss der Nutzer beim Entwickeln eines Systems das Modell seines kognitiven Prozesses mit dem kognitiven Prozess der Entwickler*innen, der in der Low-Code-Plattform implementiert ist, koordinieren. Dies schafft zwei Probleme:

1) Die Nutzenden müssen sich in den kognitiven Prozess der Entwickelnden vertiefen (die grundlegenden Prinzipien der Systementwicklung studieren), was die Nutzung von Low-Code-Plattformen für einen breiten Nutzerkreis unzugänglich macht;
2) Bei Änderungen im kognitiven Prozess der Nutzenden (Änderung der Logik zur Lösung von Geschäftsproblemen) ist es notwendig, das System neu zu erstellen oder zu ändern, während ein Verfahren durchgeführt wird, das konsistent mit den kognitiven Prozessen (eigenen und der Entwickelnden) ist.

Um diese Probleme zu lösen, schlägt die Studie einen neuen Ansatz zur Implementierung intelligenter Low-Code-Plattformen vor. Der neue Ansatz ermöglicht es Nutzenden, Formen der Logik für die Problemlösung zu konstruieren, und den Entwickelnden,

diese konstruierten Logikformen in eine Methodik zum Aufbau von Systemen umzuwandeln. Dieser Ansatz sieht vor, dass die Struktur der Low-Code-Plattform zwei Ebenen haben sollte:

- **Die intellektuelle Ebene der Nutzenden**: Auf dieser Ebene wird die Aufgabe gelöst, formell die Formen der Logik für die Lösung spezifischer Probleme des Fachgebiets zu gestalten. Dies geschieht unter Verwendung formaler und angewandter Werkzeuge der künstlichen Intelligenz.
- **Die fachliche Ebene der Entwickelnden**: Hier wandeln die Entwickelnden die Formen der Logik in eine Methodik zum Aufbau von Systemen um.

Als Fallstudie wird der Prozess der Konstruktion der Logikformen der Nutzenden für die Umwandlung von Primärdokumenten in Ausgabedokumente betrachtet.

Das in dem Paper beschriebene Vorgehen liefert Einblicke in die Herausforderung bei der Entwicklung von Low-Code-Anwendung und liefert einen Vorschlag, wie man die Low-Code-Entwicklung in der Organisation verankern kann. Der vorgeschlagene Ansatz sowie die identifizierten Herausforderungen werden bei der Erarbeitung der Low-Code-Regeln berücksichtigt.

"Towards a Governance of Low-Code Development Platforms Using the Example of Microsoft PowerPlatform in a Multinational Company" (HEUER ET AL. 2022)

Die Studie untersucht die Governance von Low-Code-Entwicklungsplattformen, insbesondere anhand des Beispiels der Microsoft PowerPlatform in einem multinationalen Unternehmen. Die Studie erforscht die Herausforderungen und Lösungsansätze zur Implementierung einer effektiven Governance-Struktur für Low-Code-Plattformen und betont die Bedeutung von Governance im Kontext des End-User Computings. Sie zielt darauf ab, Designprinzipien für die Entwicklung einer Governance für Low-Code-Plattformen zu entwickeln und zu evaluieren, um evidenzbasiertes Designwissen für die Entwicklung solcher Governance-Strukturen bereitzustellen.

Die identifizierten Herausforderungen im Zusammenhang mit Low-Code-Entwicklungsplattformen werden in Tabelle 9 dargestellt.

Tabelle 9: Herausforderungen von Low-Code-Plattformen (s. HEUER ET AL. 2022)

Herausforderungen von Low-Code-Plattformen
Drittanbieter können Erweiterungen bereitstellen
Wenig Kodierung – einfacher Zugang für nicht-technische Benutzer
Schneller Zugriff und Handhabung von (sensiblen) Daten
Zugriffsprobleme
Schnelle Umsetzung von Anforderungen
Gefahr der "Schatten-IT"

Diese Herausforderungen zeigen die Notwendigkeit einer effektiven Governance für LCDPs, um Risiken zu minimieren und den vollen Nutzen der Plattformen zu realisieren. Anschließend werden basierend auf dem Vorgehen von WEILL U. ROSS den Entscheidungsbereichen Governance Archetypen zugeordnet (siehe Tabelle 10):

Tabelle 10: Governance Archetypen für Low-Code-Plattformen (s. HEUER ET AL. 2022)

Archetypen	**Eingabe**	**Entscheidung**
Prinzipien	Föderal	Föderal
Kritische Anwendungen	Föderal	Feudal
Zukünftige Themen	Föderal	Föderal
Architektur	Föderal	IT-Monarchie
Anwendungsentwicklung	Föderal	IT-Monarchie

Im Paper wird betont, dass diese Entscheidungsbereiche wichtig für die IT-Governance sind und die Verantwortung und Unterstützung von Anwendungen, die technische Implementierung der Governance-Prinzipien und die Berücksichtigung zukünftiger Änderungen und Anforderungen behandeln. Darüber hinaus wird darauf hingewiesen, dass das Thema Low-Code derzeit in der Literatur noch sehr spärlich behandelt wird. Obwohl einige Fallstudien dokumentiert haben, wie Low-Code-Plattformen die Anwendungsentwicklung unterstützen können, gibt es kaum direkte vergleichende Literatur zu Low-Code-Governance Ansätzen. Obwohl dieses Paper einen ersten Einblick und eine Reflexion sowie eine kritische Untersuchung eines bisher wenig beachteten und nur wenig untersuchten Themas bietet, stellt es nur einen Standpunkt dar und bedarf weiterer Untersuchungen. Der Umfang dieser Forschung ist begrenzt, und die Entwicklungen und Merkmale, die im Unternehmen gefunden wurden, sind nicht notwendigerweise auf andere Unternehmen übertragbar oder sogar verallgemeinerbar. Das Paper schlägt weiteren Forschungsbedarf bezüglich der Governance-Konzepte von Plattformen, strukturellen Entscheidungen, Rollen oder Prozessen, dem Einfluss von Unternehmensstrukturen und Mitarbeiterkulturen sowie deren Beziehung zur allgemeinen Entwicklung der Mitarbeiter*innen vor.

Die in diesem Paper gewonnenen Erkenntnisse finden insbesondere Anwendung bei der Ausarbeitung von Regeln für Low-Code. Die von Heuer et al. verfolgte Methode, sich an einem existierenden Governance-Framework zu orientieren und dieses speziell für Low-Code zu adaptieren, wird in dieser Arbeit aufgegriffen und angewendet.

"Two Perspectives of Low-Code Development Platform Challenges – An Exploratory Study" (PRINZ ET AL. 2022)

Die Studie analysiert Herausforderungen bei der Verwendung von Low-Code-Plattformen aus wissenschaftlicher und praktischer Perspektive. Sie nutzt Literaturstudien, Expert*inneninterviews und praktische Studien zur Identifikation und Analyse von Herausforderungen, die in einem sozio-technischen Systemmodell kategorisiert werden.

Das sozio-technische Systemmodell betrachtet Organisationen und Technologien als ineinandergreifende Systeme. Es umfasst soziale Aspekte wie Teamstrukturen, Kultur und Prozesse sowie technische Aspekte wie Software, Hardware und Prozesse. Das Ziel dieses Modells ist es, ein besseres Verständnis dafür zu entwickeln, wie technologische und soziale Faktoren zusammenwirken. Im Kontext von Low-Code-Plattformen bezieht sich dies darauf, wie technische Tools und soziale Dynamiken sich auf die Entwicklung und Implementierung von Software auswirken. Das Modell unterstützt bei der Identifikation und Lösung von Herausforderungen, die durch das Zusammenspiel von Menschen und Technologie entstehen. Dazu wird das Sozio-technische Systemmodell von LEAVITT verwendet. Das Systemmodell besteht aus den vier Komponenten Technologie, Aufgabe, Akteur und Struktur, während Technologie und Aufgabe dem technischen System und Akteur und Struktur dem sozialen System zugeordnet werden.

Die Ergebnisse von PRINZ ET AL. zeigen, dass sowohl die wissenschaftliche als auch die praktische Gemeinschaft gemeinsame Herausforderungen erkennen, insbesondere im Bereich des Wissenstransfers, aber auch Unterschiede in Bezug auf technologische und soziale Aspekte.

Im Bereich der Strukturkomponente identifiziert die Studie von PRINZ ET AL. verschiedene Herausforderungen bei der Implementierung und Anwendung von Low-Code-Entwicklungsplattformen (LCDPs). Forscher sprechen die Herausforderungen der Strukturkomponente allgemein an. Für Organisationen besteht die Herausforderung darin, ein ausreichendes Governance-Rahmenwerk und eine Richtlinie zu etablieren, die regelt, wie die Implementierung und Anwendung der jeweiligen Plattform verwaltet wird. Praktiker hingegen thematisieren Herausforderungen der Strukturkomponente detaillierter, insbesondere im Hinblick auf Prozesse. Einige Befragte erwähnen, dass es eine Herausforderung darstellt, Entscheidungsprozesse zu etablieren. In diesem Zusammenhang ist es wichtig zu klären, welche Prozesse durchgeführt werden müssen, um eine erfolgreiche Implementierung und Anwendung der jeweiligen LCDPs zu gewährleisten. Es ist zunächst schwierig, die relevanten Prozesse zu identifizieren, die digitalisiert werden müssen.

Die Studie vergleicht acht gemeinsame Kategorien von Herausforderungen, indem sie diese in vier Quartale aufteilt, basierend auf der Häufigkeit ihrer Erwähnung aus wissenschaftlicher und praktischer Sicht. Die Achsen der resultierenden Matrix reichen von „am wenigsten erwähnt" bis „am meisten erwähnt". Darüber hinaus werden bestehende Forschungsansätze konsultiert und ein Ausblick auf zukünftige Forschung gegeben. Abbildung 2 in der Studie zeigt einen Überblick über die Herausforderungskategorien aus beiden Perspektiven. Die Studie zeigt insbesondere ein wissenschaftliches Defizit in der Erforschung einer Low-Code-Governance, obwohl diese in der Praxis schon weit verbreitet ist (siehe Abbildung 3-7).

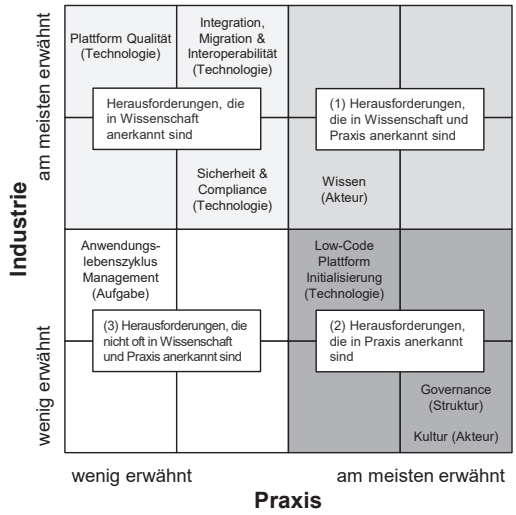

Abbildung 3-7: Überblick über Herausforderungskategorien aus Wissenschaft und Praxis (s. PRINZ ET AL. 2022)

Das Paper unterstreicht die Notwendigkeit weiterer Forschungen im Bereich der Low-Code-Governance. Zudem liefert es wertvolle Einblicke in gegenwärtige Risiken und Herausforderungen im Zusammenhang mit Low-Code, die bei der Entwicklung des Low-Code-Regelwerks Berücksichtigung finden. Die Methode, Low-Code mittels des soziotechnischen Systemmodells von Bostrom und Heinen zu strukturieren, wird ebenfalls in dieser Arbeit angewandt.

Aufgrund der geringen Anzahl an relevanten wissenschaftlichen Arbeiten zu Low-Code-Anwendungsfällen und deren Verankerung in der Organisationsstruktur werden weitere Arbeiten rund um das Management von Informationssystem-Architekturen betrachtet.

3.2.2 Informationssystem-Architekturen

Im Folgenden werden Modelle zu Informationssystem-Architekturen beschrieben, deren Erkenntnisse in dieser Arbeit berücksichtigt wurden. Die beschriebenen Modelle werden auf Erkenntnisse hinsichtlich möglicher Merkmale für die Beschreibung und Typisierung für Low-Code-Anwendungsfälle untersucht.

McFarlan's Strategic Grid (MCFARLAN ET AL. 1983)

In „McFarlan's Strategic Grid" wird die Position von Informationssystemen in verschiedenen Unternehmenstypen detailliert betrachtet. Die Autor*innen klassifizieren Unternehmen anhand der strategischen Bedeutung ihrer Informationssysteme in vier Kategorien: Strategisch, Umbruch (Turnaround), Fabrik und Unterstützung (Support). Diese Klassifizierung hilft zu verstehen, wie Informationssysteme in verschiedenen

organisatorischen Kontexten eingesetzt werden und welche Auswirkungen dies auf die Planung und Umsetzung von Informationssystem-Strategien hat. Abbildung 3-8 zeigt die vier Kategorien, die im Folgenden detaillierter beschrieben werden.

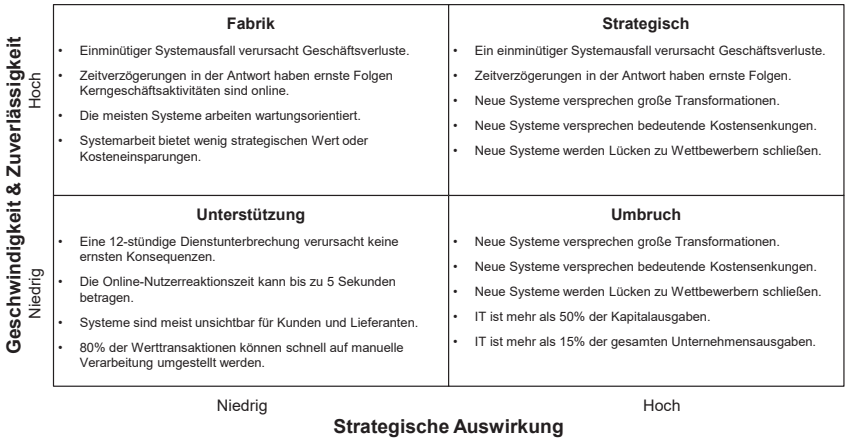

Abbildung 3-8: Informationssystemkategorien (s. MCFARLAN ET AL. 1983)

- **Strategisch (Strategic):** In diesen Unternehmen sind die täglichen Operationen und die Wettbewerbsfähigkeit stark von effizienten und innovativen Informationssystemen abhängig. Banken und Versicherungsunternehmen sind typische Beispiele, wo Informationssysteme entscheidend für den Betrieb und für die Entwicklung neuer Produkte oder Dienstleistungen sind.
- **Umbruch (Turnaround)**: Unternehmen in dieser Kategorie sind nicht vollständig von Informationssystemen für ihre täglichen Operationen abhängig, planen aber strategische Informationssystem-Projekte, die für die Erreichung ihrer langfristigen Ziele kritisch sind. Diese Projekte können dazu dienen, bestehende Geschäftsmodelle zu transformieren oder neue Wettbewerbsvorteile zu schaffen. Das Engagement für Informationssystem-Planung ist hier ebenfalls hoch.
- **Fabrik (Factory):** In dieser Kategorie sind Unternehmen, die stark auf Informationssysteme für ihre täglichen Operationen angewiesen sind, deren Informationssystem-Projekte aber eher wartungsorientiert sind und nicht unmittelbar strategische Bedeutung beinhalten. Die Informationssystem-Planung hat einen operativen Charakter, der sich auf Effizienz und Kostenkontrolle fokussiert. Unternehmen in der Fertigungsindustrie, Fluggesellschaften und einige Einzelhändler können hier eingeordnet werden.
- **Unterstützung (Support)**: Diese Unternehmen haben große Informationssystem-Budgets, sind aber weder für den laufenden Betrieb noch für die strategische Ausrichtung stark von Informationssystemen abhängig. Die Informationssystem-Abteilung spielt eine unterstützende Rolle und ist oft auf einer niedrigeren organisatorischen Ebene angesiedelt. Die Planung und strategische

Ausrichtung von Informationssystemen haben in solchen Unternehmen oft eine geringere Priorität. Beispiele können große herstellende Unternehmen sein, die trotz umfangreicher Informationssystem-Investitionen ihren Betrieb auch bei größeren Informationssystem-Ausfällen aufrechterhalten können.

Diese Kategorisierung macht deutlich, dass die Rolle und strategische Relevanz von Informationssystemen wesentlich vom spezifischen Unternehmenskontext abhängen. In dieser Arbeit wird die Differenzierung der strategischen Bedeutung von Informationssystemen auf die strategische Wichtigkeit verschiedener Anwendungen innerhalb eines Unternehmens übertragen. Diese Übertragung ermöglicht es, die strategische Relevanz als ein Merkmal für Low-Code-Anwendungsfälle zu definieren (siehe Kapitel 5.2.1). McFarlan's Strategic Grid Modell unterstreicht, dass Anwendungen mit unterschiedlicher strategischer Bedeutung in Unternehmen differenziert behandelt werden.

Dimensionen der IT-Architekturentscheidung (HOLTSCHKE ET AL. 2009)

Die Analyse der Dimensionen der IT-Architekturentscheidung, wie sie HOLTSCHKE ET AL. vornehmen, bietet einen fundierten Rahmen für Unternehmen, um ihre IT-Architektur strategisch zu planen und zu entwickeln. Durch die Berücksichtigung des Grades der Verschiedenheit der Anforderungen der Fachbereiche (homogen vs. heterogen) und der Autonomie der Fachbereiche in Bezug auf ihre IT-Architekturentscheidungen ermöglicht dieser Ansatz eine differenzierte Betrachtung der IT-Landschaft. Die daraus resultierende Matrixstruktur und die vier abgeleiteten Modelle (zentrales, unabhängiges, harmonisiertes und hybrides Modell) bieten Unternehmen die Möglichkeit, eine IT-Architektur zu gestalten, die sowohl den unternehmensspezifischen Anforderungen als auch den Zielen der Flexibilität und Effizienz gerecht wird (siehe Abbildung 3-9).

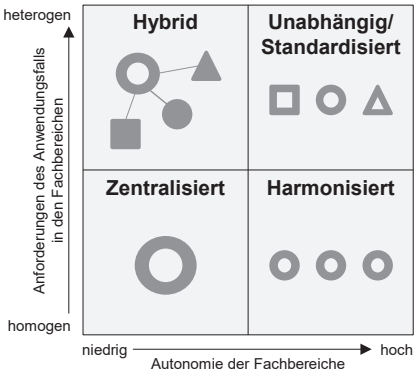

Abbildung 3-9: Dimensionen der IT-Architekturentscheidungen (s. HOLTSCHKE ET AL. 2009, S. 117-118)

Die vier beschriebenen Modelle führen zu unterschiedlichen Architekturansätzen in Unternehmen, die sich wie folgt charakterisieren lassen:

- **Zentrales Modell**: Dieser Ansatz ist besonders geeignet für Unternehmen mit einem hohen Grad an Standardisierung und Integration über verschiedene Geschäftseinheiten hinweg. Die zentrale Steuerung der IT-Architektur gewährleistet Konsistenz und ermöglicht es, Synergien zu realisieren und Kosten zu senken. Allerdings kann die geringe Entscheidungsautonomie der Geschäftseinheiten zu Flexibilitätsverlusten führen.
- **Unabhängiges Modell**: Das unabhängige Modell bietet den Geschäftseinheiten maximale Flexibilität und erlaubt es ihnen, IT-Architekturen zu entwickeln, die ihren spezifischen Bedürfnissen entsprechen. Dieses Modell eignet sich für diversifizierte Unternehmen, deren Geschäftseinheiten in sehr unterschiedlichen Märkten agieren. Die Herausforderung besteht darin, trotz der Unabhängigkeit ein Mindestmaß an unternehmensweiten Standards zu etablieren, um Kompatibilitätsprobleme zu vermeiden.
- **Harmonisiertes Modell**: Das harmonisierte Modell stellt einen Kompromiss zwischen zentraler Steuerung und dezentraler Autonomie dar. Es zielt darauf ab, ein Gleichgewicht zwischen den Vorteilen standardisierter Prozesse und der Notwendigkeit individueller Anpassungen zu finden. Ein solches Modell kann die Zusammenarbeit und Effizienz innerhalb des Unternehmens fördern, erfordert jedoch ein hohes Maß an Koordination und Abstimmung zwischen den Geschäftseinheiten.
- **Hybrides Modell:** Das hybride Modell kombiniert Elemente der zentralen und der unabhängigen Ansätze und bietet somit eine flexible Lösung für Unternehmen, die sowohl standardisierte als auch individuell angepasste IT-Lösungen benötigen. Dieser Ansatz kann besonders vorteilhaft sein, wenn bestimmte Kernprozesse unternehmensweit harmonisiert werden sollen, während gleichzeitig Raum für spezifische Anforderungen der Geschäftseinheiten bleibt.

Die Dimensionen von IT-Architekturentscheidungen zeigen, dass abhängig von der Unternehmensstrategie, den spezifischen Anforderungen und der bestehenden IT-Landschaft verschiedene Grade der Standardisierung sowie unterschiedliche Formen der Zusammenarbeit zwischen den Fachbereichen und der IT notwendig sind. Sowohl der Aspekt der Standardisierung als auch die Art der Kollaboration zwischen IT und Fachbereichen werden in dieser Arbeit thematisiert und bei der Kategorisierung verschiedener Low-Code-Anwendungsfälle berücksichtigt.

Pace-Layer Application Strategy (MESAGLIO U. HOTLE 2016)

Die „Pace-Layered Application Strategy" von Gartner bietet einen strategischen Rahmen für die Anwendungsentwicklung innerhalb von IT-Organisationen. Sie führt das Konzept der Schichtung von Anwendungen nach ihrer Änderungsrate ein: Systeme der Aufzeichnung (Systems of Record), Systeme der Differenzierung (Systems of Differentiation) und Systeme der Innovation (Systems of Innovation). Diese Strategie unterstützt Organisationen dabei, das Bedürfnis nach schneller Anpassung in bestimmten Geschäftsbereichen mit dem Bedarf an Stabilität in anderen zu vereinen. Abbildung 3-10 zeigt die Schichtung der Anwendungen.

Abbildung 3-10: Geschwindigkeitsschichten von Anwendungen (s. MESAGLIO U. HOTLE 2016)

Die Autor*innen schlagen ein „Hub-and-Spoke-Modell“ als optimale Organisationsstruktur für die meisten Anwendungsteams vor, die nach dem Prinzip der Pace-Layered-Strategie arbeiten. Der Hub konzentriert sich auf die Systeme der Aufzeichnung, die die Kerntransaktionsverarbeitung unterstützen und die kritischen Stammdaten der Organisation verwalten. Diese Systeme haben aufgrund gut etablierter Prozesse eine niedrige Änderungsrate. Die Systeme der Differenzierung und Innovation agieren als Speichen, die mehr Agilität und eine engere Ausrichtung auf Geschäftsbedürfnisse erfordern. Diese Systeme ermöglichen unternehmensspezifische Prozesse, branchenspezifische Fähigkeiten und adressieren neue Geschäftsanforderungen oder -chancen mit kürzeren Lebenszyklen und schneller Entwicklung.

Die Analyse beinhaltet auch eine Betrachtung der Governance-Unterschiede zwischen den Schichten und unterstreicht die Notwendigkeit unterschiedlicher Management- und Entwicklungsansätze für jede Kategorie. Während Systeme der Aufzeichnung Kostenersparnis und Effizienz priorisieren, fokussieren sich Systeme der Differenzierung auf Konfigurierbarkeit und Reaktionsfähigkeit auf sich ändernde Geschäftspraktiken. Systeme der Innovation hingegen priorisieren schnelle, ad-hoc-Entwicklung, um neue Geschäftschancen zu nutzen.

Zusammengefasst empfiehlt das Modell unterschiedliche Governance-Ansätze für verschiedene Arten von Anwendungen. Es wird vorgeschlagen, dass Anwendungen, die kritische Stammdaten beinhalten, zentral verwaltet werden, während Systeme mit kürzeren Lebenszyklen und schnellen Anpassungsbedürfnissen dezentraler organisiert werden können. Sowohl die Kritikalität der Daten innerhalb von Anwendungen als auch die Lebensdauer der Anwendungen werden in dieser Arbeit als Merkmale für die Typisierung von Anwendungsfällen berücksichtigt.

3.2.3 IT-Governance Modelle

Nachdem zuvor verschiedene Informationssystem-Architekturen mit Hinblick auf mögliche Typisierungsansätze für Low-Code-Anwendungsfälle untersucht wurden, werden

im Folgenden die weitverbreitetsten IT-Governance Modelle vorgestellt und deren Relevanz für diese Arbeit wird beleuchtet.

CISR (WEILL U. ROSS 2004)

Das Modell von Peter Weill und Jeanne W. Ross in ihrem Buch „IT Governance: How Top Performers Manage IT Decision Rights for Superior Results" bietet einen umfassenden Rahmen für die strategische Planung und Verwaltung von IT-Ressourcen in Unternehmen. Sie identifizieren fünf Schlüsselbereiche, die für IT-Governance-Entscheidungen kritisch sind: IT-Prinzipien, IT-Infrastrukturstrategien, IT-Architektur, IT-Anwendungsbedarf sowie IT-Investitionen und Priorisierung. Diese Bereiche bilden das Fundament für eine effektive IT-Governance, die IT-Entscheidungen so ausrichten, dass sie die Geschäftsstrategie unterstützen und zur Erreichung der Unternehmensziele beitragen.

Weill und Ross beschreiben zudem verschiedene Governance-Modelle oder Archetypen, basierend auf der Zentralisierung oder Dezentralisierung von Entscheidungsrechten:

- **Geschäftsbereichs-Monarchie**: Hier werden Entscheidungen von den Geschäftsleitern getroffen, was eine flexible Anpassung an spezifische Marktbedürfnisse ermöglicht.
- **IT-Monarchie**: Die Entscheidungsrechte sind innerhalb der IT-Abteilung zentralisiert, was eine konsistente IT-Strategie fördern kann, aber auch das Risiko birgt, dass die IT-Strategie nicht vollständig mit den Geschäftszielen übereinstimmt.
- **Föderale Modelle**: Diese kombinieren zentrale und dezentrale Entscheidungsinstanzen und erlauben eine Balance zwischen unternehmensweiten Standards und lokaler Anpassungsfähigkeit.
- **IT-Duopol**: Beschreibt eine Partnerschaft zwischen IT-Führungskräften und einem anderen Stakeholder, wie Geschäftsleitern, und fördert die Zusammenarbeit.
- **Anarchie**: Bietet maximale Flexibilität durch wenig formalisierte Strukturen für IT-Entscheidungen, birgt aber das Risiko von Inkonsistenzen und mangelnder Ausrichtung auf die Unternehmensziele.

Die Autor*innen entwickelten eine Matrix (siehe Abbildung 3-11), die IT-relevante Entscheidungsdomänen mit den in Betracht kommenden Governance-Typen in Beziehung setzt.

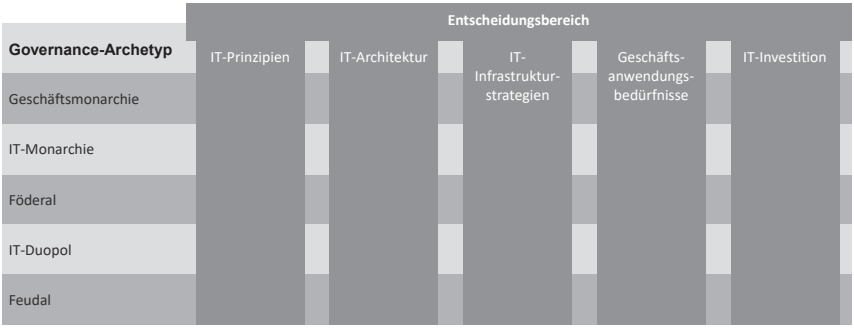

Abbildung 3-11: IT-Governance-Matrix (s. WEILL U. ROSS 2004)

Diese Matrix verdeutlicht, dass es typische Verteilungsmuster gibt, die mit der Unternehmensstrategie und -phase korrelieren. Zentralisierte Governance-Typen finden sich oft bei Firmen mit maximaler Profitrate, dezentrale Entscheidungsstrukturen bei Unternehmen im Wachstum, und ein durchmischter Ansatz bei Unternehmen, die ihre Betriebsmittel optimiert ausnutzen wollen. Ein zentrales Ergebnis der Studie von WEILL U. ROSS ist der Hinweis an Unternehmen, ihre Rahmenbedingungen zu analysieren und unter Berücksichtigung der Erfahrungen erfolgreicher, ähnlich aufgestellter Unternehmen ihre IT-Governance zu definieren, zu überarbeiten und umzusetzen. Dies unterstreicht die strategische Bedeutung von IT-Entscheidungen und bietet einen praktischen Rahmen für Organisationen, ihre IT-Governance zu verbessern, um bessere Geschäftsergebnisse zu erzielen (s. RÜTER ET AL. 2010, S. 22).

Obwohl das Governance-Modell Ansätze dafür bietet, wie IT und Fachbereiche zusammenarbeiten können, ist das Modell primär strategisch ausgerichtet und bietet wenig konkrete Handlungsempfehlungen für den operativen Betrieb. Daher gestaltet sich seine Anwendung für diese Arbeit als herausfordernd, da der Fokus dieser Arbeit speziell auf der operativen Unterstützung bei der Implementierung und dem Betrieb von Low-Code-Anwendung in Unternehmen liegt.

COBIT-Kernmodell (ISACA 2020)

Das COBIT-Kernmodell bietet einen umfassenden Rahmen für die Governance und das Management von Unternehmens-IT und ist darauf ausgerichtet, Unternehmen zu einem optimalen Betrieb zu verhelfen. Durch die Implementierung der 40 definierten Governance- und Managementziele können Unternehmen den maximalen Nutzen aus ihrem IT-Einsatz ziehen. Diese Ziele, die untereinander gleichwertig sind und als normative Vorgabe dienen, lassen sich durch Designfaktoren an den spezifischen Unternehmenskontext anpassen, priorisieren oder sogar ausschließen. Die Ziele sind in fünf unterschiedlichen Governance-Domänen verteilt (siehe Abbildung 3-12).

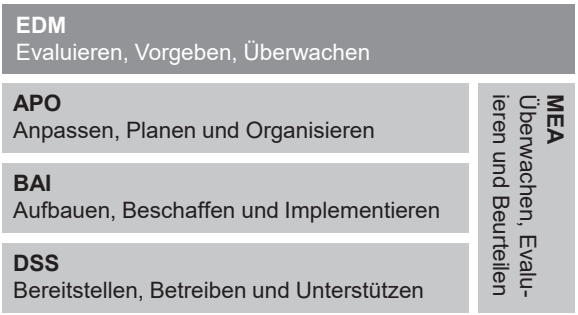

Abbildung 3-12: COBIT-Governance-Domänen (s. ISACA 2020)

Die Governance-Domänen werden im Folgenden näher beschrieben.

- **EDM (Evaluate, Direct and Monitor):** Diese Governance-Domäne schafft den Rahmen, innerhalb dessen die übrigen vier Management-Domänen agieren. Sie befasst sich mit der Bewertung, Steuerung und Überwachung der IT-Governance-Struktur eines Unternehmens, um sicherzustellen, dass die IT-Aktivitäten und -Ziele im Einklang mit den Unternehmenszielen stehen.
- **APO (Align, Plan and Organize):** Die APO-Domäne umfasst alle Ziele, die sich auf die Planung der IT-Strategie beziehen. Sie zielt darauf ab, sicherzustellen, dass die IT-Strategie und -Planung mit den Geschäftszielen abgestimmt sind und dass die IT-Organisation effektiv organisiert ist, um diese Strategien zu unterstützen.
- **BAI (Build, Acquire and Implement):** In dieser Domäne werden die Ziele zusammengefasst, die notwendig sind, um die IT-Strategie und -Pläne in greifbare Lösungen und Services umzusetzen. Es geht um den Aufbau, die Beschaffung und die Implementierung von IT-Lösungen, die den Anforderungen des Unternehmens entsprechen.
- **DSS (Deliver, Service and Support):** Die DSS-Domäne beinhaltet Ziele, die sich auf die eigentliche Bereitstellung und Unterstützung von IT-Services konzentrieren. Hierbei geht es um den Betrieb und die kontinuierliche Unterstützung der IT-Services, um den laufenden Betrieb und die Servicequalität sicherzustellen.
- **MEA (Monitor, Evaluate and Assess):** Diese Domäne dient als internes Kontrollsystem, um die Einhaltung aller Anforderungen zu überwachen und zu bewerten. Sie umfasst Ziele zur Überwachung der Leistung und Effektivität der IT-Governance und des Managements sowie zur Bewertung interner Kontrollen und Compliance.

Das COBIT-Framework ermöglicht es Unternehmen, eine strukturierte und effektive IT-Governance und ein IT-Management zu etablieren, das die Geschäftsstrategie unterstützt und zur Erreichung der Geschäftsziele beiträgt. Die klare Strukturierung in fünf Domänen erlaubt einen fokussierten Blick auf verschiedene Aspekte der IT-

Governance und die systematische Verbesserung der IT-Leistung. Obwohl das Modell mit seinen 40 Management- und Governance-Zielen eine umfassende Übersicht über alle möglichen IT-Kontrollaktivitäten in Zusammenarbeit mit den Geschäftsbereichen bietet, kann seine Komplexität eine Herausforderung darstellen und erfordert eine Anpassung an spezifische Kontexte.

ITIL (AXELOS 2021)

ITIL ist ein Framework, das auf die Optimierung des IT-Service-Managements (ITSM) ausgerichtet ist und vom Office of Government Commerce (OGC), ehemals bekannt als Central Computer and Telecommunications Agency (CCTA), entwickelt wurde. ITIL zielt darauf ab, eine strukturierte Umgebung zu schaffen, in der IT-Dienstleistungen effizient gestaltet, überführt, betrieben und kontinuierlich verbessert werden können, um die Geschäftsziele zu unterstützen. Der Schwerpunkt liegt auf der Bereitstellung qualitativ hochwertiger IT-Services, was durch ein umfassendes Set von Best Practices erreicht wird. Hierbei liegt der Schwerpunkt im Vergleich zu COBIT nicht auf der Kontrolle, sondern auf dem Service Management (s. RÜTER ET AL. 2010, S. 24). Die Best Practices sind in den Kernteilen des ITIL-Frameworks organisiert. Abbildung 3-13 zeigt eine Übersicht des ITIL-Frameworks.

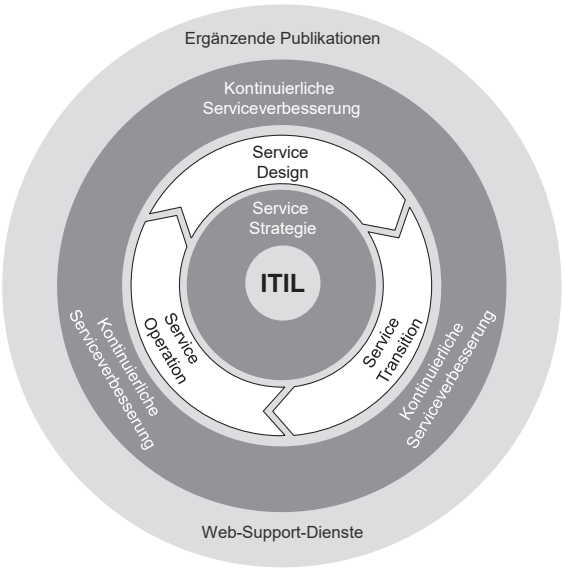

Abbildung 3-13: ITIL-Framework (s. AXELOS 2021)

Der Kernteil besteht aus Service-Strategie, Service-Design, Service-Transition, Service-Operation und kontinuierlicher Serviceverbesserung, welche im Folgenden genauer beschrieben werden.

- **Service-Strategie**: Der Grundstein, um Wert mit IT-Services zu schaffen, indem sie mit den Geschäftszielen und Kundenbedürfnissen abgestimmt werden.

Diese Phase legt die Richtung für das Management und die Bereitstellung von Services fest und sorgt dafür, dass jede Entscheidung den Unternehmenszielen dient.

- **Service-Design**: Der Prozess Services zu entwerfen, die genau auf die Geschäftsanforderungen zugeschnitten sind. In dieser Phase werden die Services so gestaltet, dass sie effektiv, effizient und skalierbar sind, um den Anforderungen des Unternehmens gerecht zu werden.
- **Service-Transition**: Gewährleistet eine reibungslose Einführung und Implementierung von Services durch sorgfältiges Änderungsmanagement und Qualitätskontrolle. Diese Phase umfasst alles, was notwendig ist, um neue oder geänderte Services sicher in den Live-Betrieb zu überführen.
- **Service-Operation**: Betrifft den täglichen Betrieb und das Management von Services. Hierbei geht es darum, sicherzustellen, dass IT-Services so effektiv und effizient wie möglich bereitgestellt werden und dass sie den Erwartungen der Nutzer entsprechen.
- **Kontinuierliche Serviceverbesserung:** Fokussiert sich auf die kontinuierliche Bewertung und Verbesserung der Qualität von IT-Services. Durch die ständige Überwachung der Leistung und die Anwendung von Verbesserungsmaßnahmen strebt diese Phase danach, den Wert der IT-Services im Laufe der Zeit kontinuierlich zu erhöhen.

Neben dem Kernteil gibt es ergänzende Veröffentlichungen und Web-Support-Dienste, die ITIL-Nutzern zusätzliche Ressourcen und Hilfestellungen bieten.

Durch die Anwendung von ITIL zielen Organisationen darauf ab, ihre IT-Prozesse klar zu strukturieren, um Effizienz zu steigern, Kundenzufriedenheit zu erhöhen und eine transparente Kommunikation zwischen allen Stakeholdern zu fördern. Die Stärke von ITIL liegt in seiner Fähigkeit, IT-Dienstleistungen mit den Geschäftszielen in Einklang zu bringen und eine kontinuierliche Verbesserung der Servicequalität zu fördern.

Eine Herausforderung bei der Implementierung von ITIL, ist der Mangel an spezifischen Anweisungen für die Eingliederung des Frameworks in bestehende Organisationsstrukturen. Dies kann für Unternehmen, die ITIL einführen möchten, besonders herausfordernd sein, da der Prozess eine erhebliche Anpassung an die spezifischen Bedürfnisse und die bestehende Kultur der Organisation erfordert. Ebenso ist ITIL auf das Service-Management und weniger auf die Anwendungsentwicklung fokussiert und bietet keine umfassenden Leitlinien für die Implementierung, was seine Anwendbarkeit in bestimmten Kontexten begrenzen kann. Daher eignet sich das Modell nur bedingt für die Ziele dieser Arbeit.

ISO/IEC 27002

Die ISO/IEC 27002, basierend auf dem British Standard BS 7799, ist ein international anerkannter Standard für das Management der Informationssicherheit. Er bietet Organisationen einen Rahmen zur Sicherung ihrer Informationen und Informationssysteme, mit einem besonderen Fokus auf die Identifikation, Bewertung und das Management von Sicherheitsrisiken, um die Vertraulichkeit, Integrität und Verfügbarkeit von

Informationen zu gewährleisten (siehe Kapitel 2.3.5). Während ISO/IEC 27002 ähnliche Ziele wie ITIL verfolgt, indem es Best Practices für das Management und die Steuerung von IT-Prozessen anbietet, liegt der Schwerpunkt ausschließlich auf der Informationssicherheit (s. RÜTER ET AL. 2010, S. 25). Der Standard umfasst eine umfangreiche Sammlung von Richtlinien und Kontrollmechanismen, die sich speziell auf die Sicherheit von Informations- und Kommunikationstechnologien konzentrieren. ITIL hingegen bietet einen breiteren Ansatz für das IT-Service-Management und adressiert eine Vielfalt an IT-Serviceprozessen einschließlich, aber nicht begrenzt auf, das Sicherheitsmanagement. Beide Frameworks ergänzen sich und können in Kombination genutzt werden, um sowohl die Effizienz von IT-Services zu verbessern als auch ein hohes Maß an Informationssicherheit zu erreichen. Viele Organisationen implementieren ITIL-Praktiken zusammen mit den Sicherheitskontrollen der ISO/IEC 27002, um einen umfassenden Ansatz für IT-Service-Management und Informationssicherheit zu verfolgen.

Im Kontext dieser Arbeit eignet sich die ISO 27002 jedoch nicht als alleiniges Governance-Modell, da sie sich spezifisch auf Informationssicherheit konzentriert und andere wichtige Aspekte, die bei der Implementierung und dem Betrieb von Low-Code-Anwendung berücksichtigt werden müssen, außer Acht lässt.

Bewertung der unterschiedlichen IT-Governance-Modelle

Die vier zuvor beschriebenen IT-Governance Modelle sind in Abbildung 3-14 zusammengefasst.

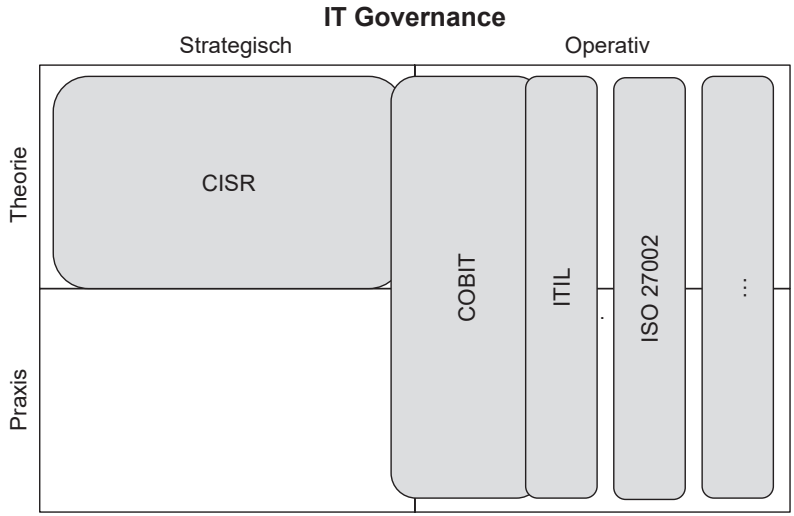

Abbildung 3-14: Einordnung bestehender IT-Governance Modelle (s. RÜTER ET AL. 2010, S. 29)

Die Differenzierung zwischen den strategischen und operativen Schwerpunkten der verschiedenen Frameworks ist für deren Anwendbarkeit in dieser Arbeit von großer Bedeutung. CISR legt den Fokus auf strategische Aspekte von Informationssystemen und liefert wichtige Einblicke in die Ausrichtung von IT und Geschäftsstrategien. Jedoch ist CISR zu strategisch orientiert, um die spezifischen operativen Herausforderungen, die mit der Implementierung und dem Betrieb von Low-Code-Anwendung einhergehen, direkt anzugehen. Obwohl ITIL und ISO 27002 wertvolle operative Handlungsempfehlungen und Best Practices für das IT-Service-Management und die Informationssicherheit bieten, erfassen sie nicht die speziellen Nuancen und Anforderungen der Low-Code-Anwendungsentwicklung und -wartung. Diese Frameworks sind für das generelle Management von IT-Services und Sicherheitspraktiken hilfreich, bieten jedoch keine maßgeschneiderten Richtlinien für Low-Code-Anwendung.

Im Gegensatz dazu präsentiert COBIT einen umfassenden Rahmen für das Management und die Governance der Unternehmens-IT, der technische Lösungen, Geschäftsziele und Risikomanagement miteinander verbindet. Die Anpassungsfähigkeit und umfassende Ausrichtung von COBIT ermöglichen dessen Anwendung auf spezifische Technologiebereiche, einschließlich der Low-Code-Entwicklung. Obwohl die Komplexität von COBIT eine Herausforderung darstellen kann, wird es in dieser Arbeit im Kapitel 6 speziell für den Kontext der Implementierung und des Betriebs von Low-Code-Anwendung zugeschnitten und so die Komplexität des Modells reduziert.

3.2.4 Zusammenfassung

Zusammenfassend ist festzustellen, dass die vorhandenen Forschungsarbeiten bezüglich einer spezifischen Governance für Low-Code-Anwendung sehr begrenzt sind. Deshalb wurde zusätzlich Low-Code unabhängige Literatur zu den Themen Informationssystem-Architekturen und IT-Governance-Modelle herangezogen. Während sich aus der Literatur zu Informationssystem-Architekturen wertvolle Erkenntnisse für die Typisierung von Low-Code-Anwendungsfällen gewinnen lassen, hat die Auseinandersetzung mit IT-Governance-Modellen das COBIT-Kernmodell als das für diese Arbeit relevanteste Framework zur Entwicklung von Regeln für Low-Code-Anwendung herausgestellt.

3.3 Schlussfolgerung und Forschungsbedarf

Diese Arbeit identifiziert eine Forschungslücke im Bereich eines spezialisierten Regelwerks für Low-Code-Anwendung in produzierenden Unternehmen. Die Auswertung vorhandener Literatur und Praxisansätze, wie in Tabelle 11 dargestellt, offenbart, dass relevante Forschungsarbeiten zu Low-Code-Anwendungsfällen und Plattformen bisher limitiert sind. Die limitierten Forschungsarbeiten machen eine Orientierung an breiter gefassten Lösungsansätzen erforderlich. Ein ganzheitliches und konsistentes Regelwerk für den Einsatz von Low-Code in der produzierenden Industrie ist bislang nicht vorhanden. Im Detail bedeutet dies:

- Der Low-Code-Forschungsbereich ist noch nicht weit erforscht. Zwar wurde in den letzten fünf Jahren einiges zu diesem Thema publiziert, die Forschung bezieht sich jedoch hauptsächlich auf die Untersuchung von Low-Code-Plattformen und der Implementierung von einzelnen Low-Code-Anwendung. Dennoch können die dort getroffenen Empfehlungen genutzt werden, um ein einfach anwendbares Modell für die Industrie abzuleiten.
- Low-Code-Anwendung werden in der produzierenden Industrie verwendet und die entwickelten Lösungen werden teilweise detailliert dargestellt. Dennoch fehlt es bisher an einer systematischen Darstellung der vielfältigen Einsatzmöglichkeiten von Low-Code sowie an einer spezifischen Typisierung von Anwendungsfällen für produzierende Unternehmen.
- Es mangelt an einem Ansatz, der ein Regelwerk speziell für Low-Code-Anwendung in produzierenden Unternehmen beschreibt. Es gibt erste Ansätze zur Entwicklung einer Low-Code-Governance, jedoch keine umfassende Beschreibung, wie ein Regelwerk aussehen könnte.
- Es wurden viele allgemeine IT-Governance Modelle für Unternehmen entwickelt, die jedoch entweder zu strategisch ausgerichtet sind oder nicht direkt auf die Anwendungsentwicklung eingehen. Da das COBIT-Kernmodell den umfassendsten Ansatz bietet, wird dieses innerhalb dieser Arbeit genutzt und auf die Low-Code-Bedingungen angepasst.

Um die beschriebenen Lücken zu schließen, werden im folgenden Kapitel 4 die Ziele dieser Arbeit vorgestellt und das methodische Vorgehen erläutert, das zur Erarbeitung der Ergebnisse führen soll.

Tabelle 11: Übersicht untersuchter Quellen zum Stand der Erkenntnisse (eigene Darstellung)

	Anwendungsdomäne		Programmiersprache		Integrationsperspektive		Unternehmensgröße		Betrachtungsraum		Modellierungszweck			Evaluation		Geltungsbereich	
	IT-Dienstleister	Produzierende Unternehmen	Programmierung	Low-Code	Organisatorisch	Technisch	KMU	Großunternehmen	Merkmale	Typisierung	Beschreibung	Erklärung	Gestaltung	Sachlogisch	Empirisch-qualitativ	Anwendungsspezifisch	Generisch
Low-Code-Anwendungsfälle																	
WANG ET AL. 2021	◑	●	●	●	◑	●	◑	◑	◑	○	◑	○	◑	●	◑	●	○
ADRIAN ET AL. 2020b	◑	●	◑	●	◑	●	◑	◑	●	○	●	○	◑	●	◑	●	○
ADRIAN ET AL. 2020a	○	●	◑	●	◑	●	○	○	○	○	●	○	○	◑	○	●	○
DALIBOR ET AL. 2022	●	●	●	●	◑	●	◑	◑	◑	○	●	○	○	●	◑	●	○
Typisierung von Low-Code-Plattformen																	
ARNOLD ET AL. 2021	◑	◑	◑	●	○	●	◑	◑	●	●	●	○	○	●	●	○	●
SZELĄGOWSKI ET AL. 2022	●	◑	◑	◑	◑	◑	●	●	◑	●	●	○	◑	●	●	○	●
IHIRWE ET AL. 2020	●	●	◑	●	○	●	◑	◑	●	●	●	○	○	●	●	○	●
SAHAY ET AL. 2020	●	◑	◑	●	○	●	○	○	●	●	●	○	○	●	●	○	●
Low-Code-Governance																	
BINZER U. WINKLER 2022	○	◑	○	●	●	○	●	●	●	○	●	◑	◑	◑	●	○	●
ROGOZOV ET AL. 2023	○	◑	◑	●	◑	◑	●	●	●	○	●	○	◑	●	●	●	○
HEUER ET AL. 2022	○	●	○	●	●	◑	○	●	○	●	◑	◑	●	●	◑	●	○
PRINZ ET AL. 2022	◑	◑	◑	●	●	●	●	●	●	○	●	○	○	◑	●	○	●
Informationssystemarchitekturen																	
MCFARLAN ET AL. 1983	○	◑	○	○	●	◑	●	●	●	●	●	◑	◑	●	◑	○	●
HOLTSCHKE ET AL. 2009	○	◑	○	○	●	◑	●	●	●	●	●	◑	◑	●	◑	○	●
MESAGLIO U. HOTLE 2016	○	◑	○	○	●	◑	●	●	●	●	●	◑	◑	●	◑	○	●

4 Herleitung des Konzeptansatzes

Dieses Kapitel bietet eine detaillierte Darstellung der methodischen Grundlagen der Modellentwicklung. Es werden sowohl inhaltliche als auch formal-konzeptionelle Anforderungen an das Regelwerk für Low-Code-Anwendung gestellt. In Kapitel 4.1 werden sie erörtert und in Kapitel 9 zur Validierung der Ergebnisse dieser Arbeit herangezogen. Diese inhaltlichen Anforderungen ergeben sich aus dem Informationsbedarf, der aufgrund der aktuellen Lücken im Stand der Erkenntnisse besteht und die formal-konzeptionellen Anforderungen zielen auf die spätere praktische Anwendung des Modells ab.

Darauffolgend werden in Kapitel 4.2 zentrale Werkzeuge für die Entwicklung von Beschreibungs- und Erklärungsmodellen sowie das abschließende Gestaltungsmodell vorgestellt. Diese Werkzeuge umfassen allgemeine Methoden zur Modellbildung, die auf der Systemtheorie basieren, sowie Vorgehensweisen der morphologischen Analyse, Typisierung und der strukturierten Inhaltsanalyse, die als Richtlinien für die Modellgestaltung fungieren. Zum Abschluss wird das spezifische Vorgehen dieser Dissertation in Kapitel 4.3 näher beschrieben.

4.1 Anforderungen an die zu entwickelnden Modelle

Die zu entwickelnden Modelle verfolgen das Ziel der Dissertation, nämlich die IT-Governance von produzierenden Unternehmen, die den Einsatz von Low-Code anstreben, um ein Regelwerk zu erweitern. Konkret soll das Ergebnis dieser Arbeit Unternehmen bei der Implementierung und bei dem Betrieb von Low-Code-Anwendung unterstützen. Um dieses Ziel zu erreichen, müssen bestimmte formal-konzeptionelle Anforderungen erfüllt werden, die insbesondere auf die Benutzerfreundlichkeit für die Anwender*innen abzielen. Diese Anforderungen werden im folgenden Abschnitt näher erläutert. Zusätzlich erfordert die inhaltliche Ausrichtung des Forschungsziels eine Aufteilung in verschiedene Teilmodelle. Die inhaltlichen Anforderungen an die Teilmodelle werden in Abschnitt 4.1.2 dargestellt.

4.1.1 Formal-konzeptionelle Anforderungen

Die praxisorientierte Forschung in den Ingenieurs- und Naturwissenschaften stellt formale Anforderungen an die Modelle, die in dieser Arbeit entwickelt werden, und bezieht sich auf ihre Anwendungsorientierung (vgl. ÖSTERLE ET AL. 2010, S. 669; SCHÜTTE 1998, S. 111-116). Diese Anforderungen lassen sich durch drei allgemeine Gütekriterien ausdrücken: Reliabilität, Validität und Utilität.

Die Validität der Modelle, also ihre allgemeine Gültigkeit und Aussagekraft, ist entscheidend und muss über die Forschungsarbeit hinaus Bestand haben. Dies ist erreicht, wenn die Modelle interdisziplinär verständlich und auf ähnliche Probleme übertragbar sind (s. NACHREINER 1997, S. 90; EISEND U. KUß 2017, S. 114). Für die Modelle dieser Dissertation bedeutet das, sie müssen erweiterbar sein, sodass sie aktuelle Anforderungen der Anwender*innen und neue Low-Code-Funktionalitäten integrieren

können, und dass das Low-Code-Regelwerk auch für Unternehmen außerhalb des produzierenden Gewerbes, beispielsweise für Versorgungsunternehmen, anwendbar sein sollte.

Die Reliabilität beschreibt die Genauigkeit und Verlässlichkeit der Modelle. Gemäß FRIEDRICHS bedeutet dies, dass die wiederholte Anwendung unter gleichen Bedingungen zu gleichen Ergebnissen führen sollte (s. FRIEDRICHS 1990, S. 102). In Bezug auf diese Arbeit muss das erste Beschreibungsmodell ein gewisses anwenderorientiertes Granularitätslevel der Anwendungsfalldefinition erfüllen und im zweiten Beschreibungsmodell dürfen keine Redundanzen innerhalb der unterschiedlichen Regeln vorliegen.

Schließlich ist die Utilität eines Modells vorrangig durch dessen Nützlichkeit, Verständlichkeit und praktische Anwendbarkeit definiert. Dies beinhaltet auch die Bewertung seiner Relevanz, Funktionalität sowie Praktikabilität und Effizienz im Einsatz (s. ÖSTERLE ET AL. 2010, S. 669). Die Relevanz ergibt sich aus dem Bedarf einer wissenschaftlichen und praktischen Lösung. Wenn die Modelle zur Bewältigung der gegebenen Problemstellung beitragen, ist die Funktionalität gegeben (s. ÖSTERLE ET AL. 2010, S. 669). In dieser Arbeit zeigt sich die Relevanz durch die Nachfrage aus der Praxis nach einer Unterstützung zur Implementierung und zum Betrieb von Low-Code-Anwendung sowie durch den Bedarf an einem wissenschaftlichen Modell für die strukturierte Gestaltung möglicher Maßnahmen für die operative Umsetzung. Die Funktionalität ist erfüllt, wenn die Arbeit eine Anleitung für die Erweiterung der IT-Governance und die Gestaltung entsprechender Maßnahmen zur Einhaltung der Regeln bietet. Diese Kriterien allein reichen jedoch nicht aus, um die Utilität des Modells für die tatsächlichen Anwender*innen zu bewerten. Deshalb müssen Gestaltungsanforderungen, wie Aspekte der Praktikabilität, eine klare Formulierung der Modellelemente, eine zielgruppenorientierte Aufbereitung von Zielen und Lösungswegen sowie die Effizienz berücksichtigt werden. In Bezug auf diese Arbeit bedeutet dies, dass sowohl die IT-Spezialist*innen als auch die IT-ferneren Fachbereiche in einem produzierenden Unternehmen die Modelle intuitiv verstehen und anwenden können. Dadurch soll deren Nutzung zu einer effizienteren Erweiterung der IT-Governance führen als herkömmliche Methoden.

4.1.2 Inhaltliche Anforderungen

Zusätzlich zu formalen Anforderungen müssen die Modelle auch inhaltlichen Anforderungen entsprechen, die sich aus den Problemstellungen und Zielsetzungen aus Kapitel 1.2 ableiten lassen.

Eine wesentliche inhaltliche Anforderung an das Modell der Low-Code-Anwendungsfälle ist die Darstellung von Merkmalen und Merkmalsausprägungen, da keine etablierte Beschreibung von Low-Code-Anwendungsfällen existiert (siehe Kapitel 3.1). Daher ist es essenziell, dass die zu bestimmenden Merkmale und Ausprägungen die vorhandenen Beschreibungen einzelner Anwendungsfälle einbeziehen und sowohl von hoher theoretischer als auch anwendungsbezogener Qualität sind. Die Anforderungen

an das erste Beschreibungsmodell gelten als erfüllt, wenn es gelingt, die zentralen Merkmale und Merkmalsausprägungen vollständig, praxistauglich und überschneidungsfrei zu erfassen (vgl. WELTER 2006, S. 114).

Ein weiteres Ziel ist die Entwicklung heterogener Anwendungsfalltypen, sodass die identifizierten Typen deutlich voneinander abgegrenzt werden können. Basierend auf den ermittelten Merkmalen und deren Ausprägungen soll ein einheitliches Verständnis der verschiedenen Low-Code-Anwendungsfälle durch Typenbildung erreicht werden. Daraus ergibt sich die Anforderung, dass die ermittelten Typen in der Realität existieren und ihre beschreibenden Merkmalskombinationen logisch kohärent, empirisch überprüfbar und praktisch anwendbar sind (s. WELTER 2006, S. 115-116).

Angesichts der fehlenden wissenschaftlichen Beiträge zu Low-Code-Governance und Regelwerken (siehe Kapitel 3.2.1) besteht die Anforderung des zweiten Beschreibungsmodells darin, ein Low-Code-Regelwerk zu beschreiben, welches den möglichen Risiken und Herausforderungen bei der Implementierung und dem Betrieb von Low-Code-Anwendung entgegenwirkt. Dabei sollten ausschließlich die relevanten und tatsächlich existenten Risiken von Low-Code adressiert werden. Dies soll die Akzeptanz und Anwendbarkeit des Regelwerks in der Praxis sicherstellen.

Zudem ist es erforderlich, die in der Praxis beobachtbaren wechselseitigen Wirkbeziehungen zwischen den ersten beiden Modellen zu dokumentieren und zu erläutern. Die Erklärung dieser Wirkbeziehungen ermöglicht es Unternehmen, die Auswirkungen verschiedener Low-Code-Anwendungsfalltypen auf die jeweiligen Low-Code-Regeln zu bewerten. Diese Wirkbeziehungen bilden zudem die Basis für die Ableitung von praxisnahen Gestaltungsempfehlungen.

Neben den spezifischen inhaltlichen Anforderungen an die Modelle werden zusätzliche Kriterien festgelegt, die die Akzeptanz und Anwendbarkeit des finalen Regelwerks steigern sollen. Eine dieser Anforderungen ist die interne Konsistenz der einzelnen Modelle, die sicherstellt, dass die Gesamtheit der Teilmodelle ein kohärentes Bild der komplexen Realität darstellt und so einen nahtlosen, verständlichen Zusammenhang innerhalb des Regelwerks für den praktischen Einsatz gewährleistet. Weiterhin wird die Visualisierung mittels grafischer Darstellungen der einzelnen Modelle gefordert, um auch verbal schwer darstellbare Inhalte klar und übersichtlich zu vermitteln (s. WELTER 2006, S. 116). Ein weiteres wichtiges Kriterium ist die Anpassungsfähigkeit des Regelwerks. In Anbetracht der kontinuierlichen Weiterentwicklung von Low-Code-Plattformen und der sich verändernden Arbeitswelt soll auch das Dissertationsvorhaben diese Dynamik berücksichtigen. Aus diesem Grund werden die Teilmodelle so gestaltet, dass sie eine entsprechende Flexibilität zur Anpassung an neue Entwicklungen bieten.

All diese inhaltlichen Anforderungen müssen den zuvor genannten formalen Anforderungen entsprechen, um den maximalen Nutzen für die Anwender*innen zu gewährleisten.

4.2 Methodische Grundlagen

Im nächsten Abschnitt wird erläutert, wie die Modellentwicklung dieser Dissertation unter Berücksichtigung der zuvor definierten Anforderungen und unter Verwendung bestehender Modelle und Methoden aus der Wissenschaft durchgeführt wurde. Der verwendete Ansatz ermöglicht so eine Maximierung der Relevanz und Anwendbarkeit des Forschungsergebnisses.

4.2.1 Systemtheorie

Die Systemtheorie wird als eine interdisziplinäre Perspektive verstanden, die Methoden und Denkmodelle für das Erklären und Beschreiben komplexer Systeme bietet (vgl. SCHWANINGER 2015, S. 4; ULRICH U. HILL 1976, S. 308). Sie abstrahiert auf der Ebene von Zuständen, Eigenschaften und Verhaltensweisen und ermöglicht es so, reale Probleme unabhängig von ihrer materiellen Existenz zu betrachten. Dies erlaubt eine simultane Analyse verschiedener Parameter. Die Gestaltung von Systemen wird als „Systems Engineering“ bezeichnet. Dieser Ansatz hat sich besonders bei der Erkennung, Beschreibung und Lösung komplexer organisatorischer Fragestellungen bewährt, da er aufgrund seiner Fähigkeit zur Abstraktion ohne mathematische Modellierungsansätze auskommt. Dies ermöglicht es, auch komplexere Systeme mit allgemeinen Hilfsmitteln der Systemtechnik darzustellen (vgl. HABERFELLNER ET AL. 2015, S. 27; ULRICH U. HILL 1976, S. 308; ZANGEMEISTER 1976, S. 23). Wichtige Elemente des System Engineerings sind dabei die systematische Vorgehensweise und das Systemdenken (s. HABERFELLNER ET AL. 2015, S. 28).

In diesem Kontext wird ein System als ein abgrenzbares, zusammengesetztes Ganzes definiert, das aus einzelnen, miteinander in Beziehung stehenden Elementen besteht (s. HABERFELLNER ET AL. 2015, S. 34; KRALLMANN ET AL. 2013, S. 34). Die Strukturierung und Abgrenzung des Systems sowie die Organisation seiner Elemente erfolgt auf Basis der Eigenschaften dieser Elemente und ihrer kausalen Verbindungen, dargestellt durch Beziehungen (s. HABERFELLNER ET AL. 2015, S. 34; KRALLMANN ET AL. 2013, S. 34). Das System wird durch die Systemgrenze von der Umwelt abgegrenzt, wobei die Umwelt als eigenes System betrachtet wird, das Einfluss auf das betrachtete System nimmt. So repräsentiert das System ein Sub- oder Untersystem, während die Umwelt als ein Übersystem fungiert, das mehrere Systeme umfasst (siehe Abbildung 4-1, HABERFELLNER ET AL. 2015, S. 36). Die Elemente werden nicht weiter unterteilt und als „Blackbox" angesehen. Im Gegensatz zu den Eingangs- und Ausgangsgrößen, die im jeweiligen Problemkontext von Bedeutung sind, ist der innere Aufbau einer „Blackbox" irrelevant (s. HABERFELLNER ET AL. 2015, S. 38-40).

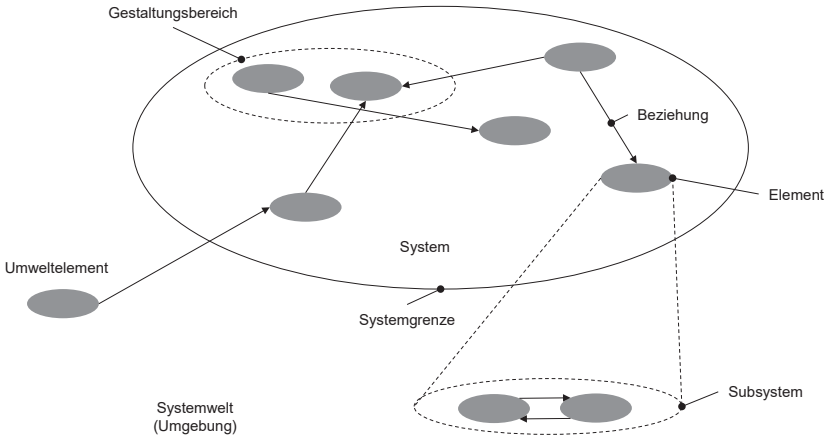

Abbildung 4-1: Systemtheorie (eigene Darstellung i. A. a. HABERFELLNER ET AL. 2015, S. 34)

Das Hauptziel der Systemtheorie ist es, Modelle zu entwickeln, die komplexe Zusammenhänge und Verknüpfungen innerhalb eines Systems sowie zwischen seinen Subsystemen beschreiben (s. HABERFELLNER ET AL. 2015, S. 41). In dieser Dissertation wird die Systemtheorie daher als Grundlage für die Modellbildung verwendet. Besonders relevant ist hierbei der Ansatz von VESTER, der betont, dass das System als Ganzes untersucht werden sollte. Dieser Ansatz findet sich in der übergeordneten Betrachtung der Anwendungsfälle von Low-Code wieder (s. VESTER 2015, S. 157; JUSSEN 2016, S. 74).

Ferner beschreibt die Theorie der sozio-technischen Systeme die Idee, dass sowohl soziale als auch technische Faktoren in Arbeitsumgebungen miteinander verflochten sind und die Leistung des Systems beeinflussen (s. PASMORE 1995). Sie sind sowohl in organisatorischen als auch in gesellschaftlichen Kontexten von Bedeutung. Somit ist eine sorgfältige Betrachtung von sozialer Dynamik und von technischen Elementen erforderlich. Sozio-technische Systeme berücksichtigen komplexe Interaktionen zwischen sozialen und technischen Elementen, um effektiv zu funktionieren (s. SUTTON U. SUTTON 1990). Leavitts Modell beschreibt die vier interdependenten Faktoren – Menschen, Aufgaben, Technologien und Strukturen – in sozio-technischen Systemen (siehe Abbildung 4-2, LEAVITT 1962).

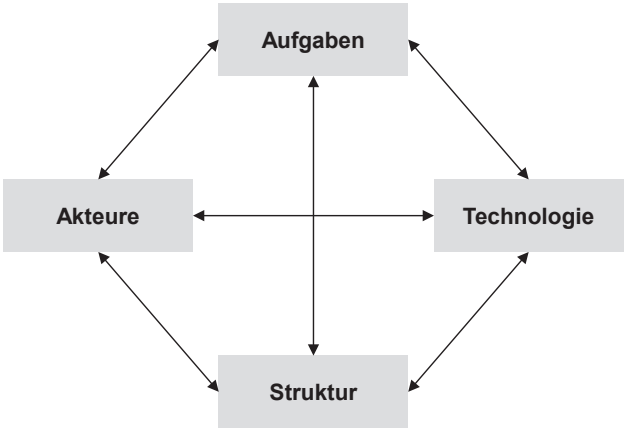

Abbildung 4-2: Schlüsselfaktoren in sozio-technischen Systemen (s. LEAVITT 1962)

Dieses Modell wird oft zur Analyse von organisatorischen Veränderungen und zur Untersuchung der Wechselwirkungen zwischen sozialen und technischen Komponenten in Organisationen verwendet (s. SUTTON U. SUTTON 1990).

In der Systemtheorie wird das zu untersuchende Problem mit fortschreitendem Verlauf immer genauer definiert: ein Prozess, der als Entwicklung vom „Groben zum Detail" bekannt ist. Dabei wird zwischen der Eingrenzung des Untersuchungs- und des Gestaltungsbereichs unterschieden. Zusätzlich wird das Prinzip der Typenbildung angewandt, welches eine Vielzahl von Untersuchungsobjekten anhand ausgewählter Kriterien systematisch einordnet. Dies stellt sicher, dass der Lösungsraum schrittweise und basierend auf bewertenden Faktoren wie Wirkung, Voraussetzungen oder Konsequenzen verkleinert wird (s. HABERFELLNER ET AL. 2015, S. 61-64). Diese methodische Herangehensweise wird in der Beschreibung der Low-Code-Anwendungsfälle und den Regeln für den Einsatz von Low-Code in den Kapiteln 5 und 6 dieser Arbeit angewandt. Durch diese strukturierte Vorgehensweise wird sichergestellt, dass das Problemfeld präzise abgesteckt und schrittweise bearbeitet wird, sodass effektive und gut durchdachte Lösungen entwickelt werden können.

4.2.2 Modellbildung

In der Wissenschaft dient die Modellierung dem Erkenntnisgewinn, indem sie eine komplexe Theorie in einem Konstrukt darstellt, das zur Verifikation oder Falsifizierung dieser Theorie genutzt werden kann. Ein Modell ist die Darstellung eines Gegenstandsbereichs in einem Zeichensystem (s. STACHOWIAK 1976, S. 56-58). STACHOWIAK hat drei Hauptmerkmale der zweckgerichteten Charakterisierung von Modellen identifiziert, die herangezogen werden (s. STRAHRINGER 2016, S. 21). Sie bestehen aus dem Abbildungsmerkmal, dem Verkürzungsmerkmal und dem pragmatischen Merkmal:

- **Abbildungsmerkmal**: Bezeichnet die grafische Darstellung des Originals mit einer angemessenen Ähnlichkeit.

- **Verkürzungsmerkmal**: Bezieht sich auf Aspekte des Originals, die in der Modellierung vernachlässigt oder mit einem hohen Grad der Abstraktion behandelt werden.
- **Pragmatisches Merkmal**: Ein zusätzliches Attribut der Abbildung des Originals, das dem Erkenntnisgewinn dient. Beispielsweise können Kontextinformationen dem Original zugeordnet werden, um dieses Merkmal zu erfüllen.

In der wissenschaftlichen Praxis haben sich je nach Untersuchungszweck verschiedene Arten von Modellen etabliert, die in Abbildung 4-3 dargestellt sind.

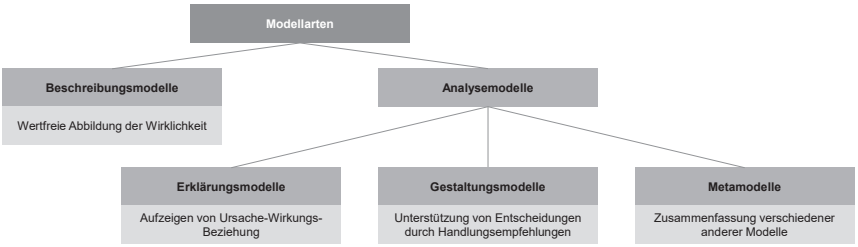

Abbildung 4-3: Modellarten (s. ZELEWSKI 2008, S. 45; SIEGERS 2016, S. 110)

In der allgemeinen Modelltheorie wird zwischen Beschreibungs- und Analysemodellen unterschieden. Während Beschreibungsmodelle eine wertfreie Abbildung der Wirklichkeit anstreben, gehen Analysemodell über die reine Beschreibung hinaus und nehmen eine weiterführende Analyse, Interpretation oder Erklärung der einzelnen Modelle vor (s. ZELEWSKI 2008, S. 45). Die Modellarten werden im Folgenden kurz erläutert.

- **Beschreibungsmodelle**: Diese Modelle zielen darauf ab, empirische Beobachtungen möglichst genau abzubilden. Sie haben eine rein deskriptive Funktion, das heißt, sie stellen Sachverhalte dar, ohne diese zu analysieren. Ihre Gestaltung hängt von der jeweiligen Aufgabe ab. Ein Beispiel für ein Beschreibungsmodell ist eine Landkarte (s. BÖHM U. FUCHS 2002, S. 26; LEHNER ET AL. 2008, S. 30-31; JUNG 2012, S. 43).
- **Erklärungsmodelle**: Aufbauend auf Beschreibungsmodellen dienen Erklärungsmodelle dazu, Zusammenhänge, Gesetzmäßigkeiten und Ursachen zu erforschen. Sie konzentrieren sich nicht auf Einzelsituationen, sondern auf Allgemeingültigkeiten über mehrere Einzelsituationen hinweg. Erklärungsmodelle haben das Ziel, Beschreibungsmodelle zu erweitern, Verhalten zu erklären oder zukünftiges Verhalten vorherzusagen. Sie sind oft hypothesengetrieben und erfordern bei einem empirischen Ansatz entsprechende Datenerhebungen zur Validierung (s. LEHNER ET AL. 2008, S. 31; JUNG 2012, S. 44).
- **Gestaltungsmodelle**: Aufbauend auf Beschreibungs- oder Erklärungsmodellen stellen Gestaltungsmodelle die höchste Klasse an Modellen dar und dienen dem Erreichen von Zielen. Sie bieten Gestaltungsalternativen an und erweitern ein System um einen Sinn, Zweck oder ein Ziel. Werden diese Modelle um die Ziele eines Entscheidungsträgers ergänzt, werden sie auch als

Entscheidungsmodelle bezeichnet (s. LEHNER ET AL. 2008, S. 31-32; ZELEWSKI 2008, S. 46). Allerdings weisen solche Modelle in ihrer Anwendung Grenzen auf, da sie soziale Zusammenhänge und menschliches Verhalten, die bei Entscheidungen in Unternehmen eine große Rolle spielen, nicht vollständig abbilden können (s. JUNG 2012, S. 45-46).

- **Metamodelle**: Meta- oder Referenzmodelle: Diese Modelle verfolgen das übergeordnete Ziel, als Blaupausen für die Entwicklung von Modellen zu dienen. Als „Modelle von Modellen" fassen sie verschiedene andere Modelle zusammen (s. LEHNER ET AL. 2008, S. 32).

Neben den zuvor genannten Modelltypen gibt es auch eine Reihe weiterer Modelle, wie Optimierungs-, Verifikations-, Steuerungs- oder Simulationsmodelle. Diese Modelle dienen eher mathematischen oder physikalischen Zwecken und finden in der vorliegenden Arbeit keine Anwendung. (s. BÖHM U. FUCHS 2002, S. 26; LEHNER ET AL. 2008, S. 32)

In dieser Arbeit werden primär Beschreibungsmodelle sowie ein Erklärungs- und ein Gestaltungsmodell entwickelt. Eine Übersicht dieser Modelle und deren Bezug zur Realwelt wird in Abbildung 4-4 dargestellt. Diese spezifische Auswahl und Anwendung von Modellen ermöglicht es, die relevanten Aspekte der Forschungsarbeit präzise zu erfassen, zu erklären und Gestaltungsempfehlungen abzuleiten, die auf den ermittelten Erkenntnissen basieren.

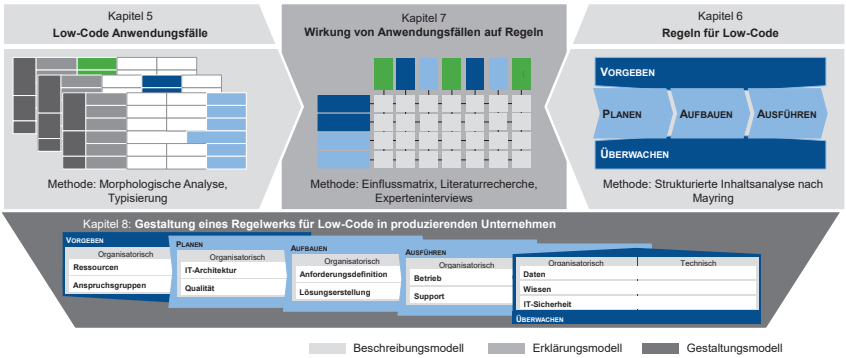

Abbildung 4-4: Übersicht der Modelle (eigene Darstellung)

4.2.3 Morphologische Analyse

Der Begriff der *Morphologie* stammt ursprünglich aus der altgriechischen Bezeichnung „morphe" und bedeutet so viel wie „Gestalt" oder „Form". Allgemein ist die Definition der Morphologie „das Studium der Form oder der Muster" (s. RITCHEY 2002, S. 2), also die Gestaltung und Anordnung der Einzelteile eines Objektes und legt fest wie diese zusammenpassen, um ein Ganzes zu bilden. Verwendung fand diese Lehre bereits verstärkt in den Natur- und Geisteswissenschaften im 18. Jahrhundert und wurde in dieser Zeit unter anderem durch *Johann Wolfang von Goethe* oder *Karl Friedrich*

Burdach geprägt (s. MAREIS 2012, S. 110). Der erste Wissenschaftler, der diese Methodik jedoch für die „Strukturierung und Untersuchung der Zusammenhänge von mehrdimensionalen und nicht quantifizierbaren Problemstellungen“ (s. RITCHEY 2002, S. 1) nutze, war der Schweizer Astrophysiker *Fritz Zwicky* im Jahr 1949. Kern von Zwickys Verfahren ist der *morphologische Kasten*. Innerhalb dieser linearen Matrix werden im Rahmen einer definierten Problemstellung vielzählige und unabhängige Parameter, als Bestimmungsgrößen oder auch Merkmale bezeichnet, ausgewählt. Diskrete Ausprägungen konkretisieren diese Merkmale und stellen die theoretisch möglichen Lösungen der Problemstellung dar. Zwicky nutze diese Methodik mitunter, um Entwicklungsmöglichkeiten für Triebwerke und Antriebsstoffe festzulegen sowie bekannte Energietransformationen abzubilden (s. MAREIS 2012, S. 114-116; SCHULTE-ZURHAUSEN 1995, S. 445).

In ZWICKYS Verfahren wird die morphologische Methode durch fünf iterative Prozessschritte charakterisiert (s. ZWICKY 1949, S. 5). Zunächst soll die zu lösende Problemstellung klar definiert und so weit wie möglich vereinfacht werden. Im zweiten Schritt werden dann alle Merkmale und ihre Ausprägungen, die für die Lösung von Bedeutung sein könnten, systematisch herausgearbeitet. Bei der Auswahl der Merkmale ist es relevant, dass diese möglichst eindeutig voneinander getrennt und unabhängig sind, damit jede Lösungsmöglichkeit am Ende der Analyse zulässig ist. Bei den Merkmalsausprägungen werden hingegen alle theoretisch denkbaren Optionen aufgezählt. Jedoch empfiehlt die Literatur (s. SCHULTE-ZURHAUSEN 1995, S. 445) auch, dass die Matrix nicht zu groß wird und die Merkmale stattdessen, wenn möglich, zu Oberbegriffen zusammengefasst werden. Im dritten Schritt wird der morphologische Kasten erstellt, der die Gesamtheit aller ermittelten Merkmale und deren Ausprägungen abbildet. Im Anschluss werden diese geprüft und hinsichtlich ihres Beitrags zur Lösung der Problemstellung bewertet. Als fünfter und letzter Schritt wird dann die optimale Lösung des morphologischen Kastens ausgewählt (s. RITCHEY 1998, S. 3), indem gedanklich oder grafisch realistische Kombinationen der Ausprägungen gebildet werden.

Tabelle 12 zeigt beispielhaft, wie ein solcher morphologischer Kasten aussehen könnte. Die linke Spalte der Tabelle listet alternative Merkmale auf, während die restlichen Spalten denkbare Ausprägungen dieser Merkmale beinhalten.

Tabelle 12: Aufbau eines morphologischen Kastens zur Typisierung von Fahrrädern (eigene Darstellung)

Merkmal	Ausprägung 1	Ausprägung 2	Ausprägung 3	…
Antrieb	Mechanisch	Elektrisch	Hybrid	...
Rahmenmaterial	Stahl	Aluminium	Carbon	…
Gänge	3	7	22	…
Bremskraft	Gering	Mittel	Hoch	
Federung	Nicht vorhanden	Vorhanden		
…	…	…	…	…

Bei der Bildung eines morphologischen Kastens unterscheidet man generell zwischen klassifizierenden Merkmalen, abstufbaren Merkmalen und Variablen, wie in Tabelle 12 exemplifiziert. Klassifizierende Merkmale weisen eine binäre Eigenschaft auf, d.h., sie sind entweder vorhanden oder nicht, wie das Beispiel der „Federung“ zeigt. Abstufbare Merkmale, repräsentiert durch die „Bremskraft“, erlauben eine Rangordnung basierend auf der Ausprägung der Merkmale, entsprechend einer ordinalen Skalierung. Variablen hingegen basieren auf einer metrischen Skala, die eine unbegrenzte Anzahl von Abstufungen mit quantifizierbaren Unterschieden ermöglicht, was durch das Merkmal „Gänge“ veranschaulicht wird (s. Fleiß 2010, S. 5-6).

Anhand der morphologischen Analyse ist es folglich möglich, einen komplexen Problembereich strukturiert in seine Teilbereiche herunterzubrechen und zu analysieren. Diese Methodik wird daher als Schlüsselelement in der vorliegenden Forschungsarbeit genutzt, um die Komplexität der Low-Code-Anwendungsfälle tiefgreifend zu durchleuchten.

4.2.4 Typisierung

Die Typisierung ist eine analytische Forschungsmethode mit dem Ziel, eine Vielzahl von Untersuchungsobjekten anhand ausgewählter Kriterien systematisch zu kategorisieren (s. Kluge 1999, S. 31; Welter 2006, S. 113). Im Gegensatz zu Schlussfolgerungsverfahren und interpretativen Methoden findet Typisierung Anwendung in verschiedenen Forschungsdisziplinen, sowohl zur Entscheidungsunterstützung als auch zur Erstellung von Systematiken (s. Große-Oetringhaus 1974; Knoblich 1969).

Typisierung berücksichtigt mehr Differenzierungsmerkmale als eine Klassifikation, meist mindestens zwei, die mehrfach abgestuft sein können (s. Knoblich 1969, S. 142-143). Ein weiterer Unterschied zur Klassifikation ist, dass Typen auch Ähnlichkeiten aufweisen können, während Klassen in der Klassifikation trennscharf sein müssen (s.

KLUGE 1999, S. 33; WELTER 2006, S. 114). Dies ermöglicht es, komplexe Realitäten zu analysieren, die für eine trennscharfe Klassifikation zu vielschichtig sind (s. ZIEGLER 1973, S. 11-13). Die Typisierung ist eine systematische, beschreibende Forschungsmethode, die als Basis für Erklärung und Gestaltung von Sachverhalten dient (s. WELTER 2006, S. 114). Sie ermöglicht eine anschauliche Analyse, indem sie unterschiedliche Merkmalsausprägungen aufzeigt, ordnet und zweckorientiert Typen zuweist (s. GROßE-OETRINGHAUS 1974, S. 52; KNOBLICH 1969, S. 143; WELTER 2006, S. 114). Die Typenbildung ist zweckbezogen und abhängig vom Untersuchungszweck (s. GROßE-OETRINGHAUS 1974, S. 52; KNOBLICH 1969, S. 143; WELTER 2006, S. 113). Die Gesamtheit aller ermittelten Typen wird Typologie genannt (vgl. GROßE-OETRINGHAUS 1974; WELTER 2006, S. 116).

Da die Entwicklung von Typologien für Informationssysteme oft unsystematisch durchgeführt wird, schlagen NICKERSON ET AL. ein strukturiertes Vorgehen zur Typologie Entwicklung von Informationssystemen vor (s. NICKERSON ET AL. 2013). Als erstes wird der Zweck der Typisierung bestimmt. Der Zweck dient als Grundlage für die Auswahl der Merkmale in der Typologie (s. NICKERSON ET AL. 2013). Als nächstes werden die Endbedingungen festgelegt. Diese umfassen objektive (siehe Tabelle 13) und subjektive (siehe Tabelle 14) Kriterien, die bestimmen, wann der Entwicklungsprozess abgeschlossen ist.

Tabelle 13: Übersicht der objektiven Endbedingungen des Iterationsprozesses (eigene Darstellung i. A. a. NICKERSON ET AL. 2013, S. 344)

Bedingung
Alle Objekte oder eine repräsentative Stichprobe von Objekten wurden untersucht.
Keine Objekte wurden in der letzten Iteration mit ähnlichen Objekten zusammengeführt oder in mehrere Objekte aufgeteilt.
Mindestens ein Objekt ist unter jeder Ausprägung jedes Merkmals klassifiziert.
Keine neuen Merkmale oder Ausprägungen wurden in der letzten Iteration hinzugefügt.
Keine Merkmale oder Ausprägungen wurden in der letzten Iteration zusammengeführt oder aufgeteilt. Jedes Merkmal ist einzigartig und nicht wiederholt (d.h. keine Merkmalsduplikation).
Jede Ausprägung ist innerhalb seines Merkmals einzigartig (d.h. keine Ausprägungsduplikation innerhalb eines Merkmals).
Jede Zelle (Kombination von Ausprägungen) ist einzigartig und wird nicht wiederholt (d.h. keine Zellduplikation).

Tabelle 14: Übersicht der subjektiven Endbedingungen des Iterationsprozesses (i. A. a. NICKERSON ET AL. 2013, S. 344)

Bedingung	Fragestellung
prägnant	Erlaubt die Anzahl der Merkmale, dass der morphologische Kasten sinnvoll erscheint, ohne schwerfällig oder überwältigend zu wirken?
robust	Ermöglichen die Merkmale und Ausprägungen eine ausreichende Differenzierung zwischen den untersuchten Objekten?
umfassend	Können alle Objekte oder eine (zufällige) Stichprobe an Objekten innerhalb des Untersuchungsbereichs beschrieben werden? Sind alle Merkmale der relevanten Objekte identifiziert?
erweiterbar	Kann ein neues Merkmal oder eine neue Ausprägung eines bestehenden Merkmals leicht hinzugefügt werden?
erklärend	Was sagen die Merkmale und Ausprägungen über das Untersuchungsobjekt aus?

Im nächsten Schritt wird der Ansatz ausgewählt und die Merkmale sowie Merkmalsausprägungen bestimmt. Hier unterscheidet man zwischen zwei grundlegenden Methoden zur Bildung oder Ableitung von Typen in der Forschung:

- **Retrograde Typenbildung:** In diesem Ansatz nutzt der Forscher sein tiefgehendes Wissen über den Untersuchungsgegenstand, um rückwärtsgerichtet (retrograd) verschiedene charakteristische Merkmalsausprägungen zu identifizieren, die spezifisch für die jeweiligen Typen sind. Der Forscher arbeitet also von einem breiten Verständnis des Feldes aus und grenzt es auf spezifische Eigenschaften ein. (s. WELTER 2006, S. 115)
- **Progressive Typenbildung:** Hier beginnt der Forscher mit der Analyse einzelner Merkmale und deren Ausprägungen. Aus dieser Detailanalyse heraus entwickelt er vorwärtsgerichtet (progressiv) Typen durch die Verknüpfung dieser Merkmale. Dieser Prozess der Typenkonstruktion muss anschließend im unternehmerischen oder realen Kontext verifiziert werden, um die Gültigkeit und Anwendbarkeit der Typisierung zu gewährleisten. (s. WELTER 2006, S. 116)

Die Kombination beider Ansätze führt zu aussagekräftigen Typen. Als letztes folgt der iterative Entwicklungsprozess. Der Entwicklungsprozess ist iterativ und umfasst die Identifizierung von Objekten, die Bestimmung ihrer gemeinsamen Merkmale und die Gruppierung dieser Merkmale zu Dimensionen. Dieser Prozess wird wiederholt, bis die Endbedingungen erfüllt sind. (s. NICKERSON ET AL. 2013)

Im Folgenden wird die Typologie eines Rennrads auf der Grundlage eines morphologischen Kastens zur Kategorisierung von Fahrrädern dargestellt (siehe Kapitel 4.2.3). In Tabelle 15 ist der Typ „Rennrad“ blau markiert und hebt die spezifischen Eigenschaften hervor, die ein Rennrad charakterisieren: ein mechanischer Antrieb, ein

Rahmen aus Carbon, eine 22-Gang-Schaltung, hohe Bremskraft und das Fehlen einer Federung.

Tabelle 15: Typologie eines Rennrads auf Basis eines morphologischen Kastens zur Typisierung von Fahrrädern (eigene Darstellung)

Merkmal	Ausprägung 1	Ausprägung 2	Ausprägung 3	…
Antrieb	Mechanisch	Elektrisch	Hybrid	...
Rahmenmaterial	Stahl	Aluminium	Carbon	…
Gänge	3	7	22	…
Bremskraft	Gering	Mittel	Hoch	
Federung	Nicht vorhanden	Vorhanden		
…	…	…	…	…

In der vorliegenden Arbeit sollen die vorgestellten Methoden der morphologischen Analyse und der Typisierung miteinander kombiniert werden, um Low-Code-Anwendungsfalltypen für produzierende Unternehmen zu entwickeln. Diese Typenbildung basiert auf der kombinierten Anwendung retrograder und progressiver Typenbildung nach WELTER und wird durch reale Beispiele verifiziert, um die Forderung nach faktischer Wahrheit zu erfüllen (s. KNOBLICH 1969, S. 32). Abbildung 4-5 zeigt das Vorgehen in vier Schritten nach NICKERSON ET AL., welches in dieser Arbeit durchgeführt wurde.

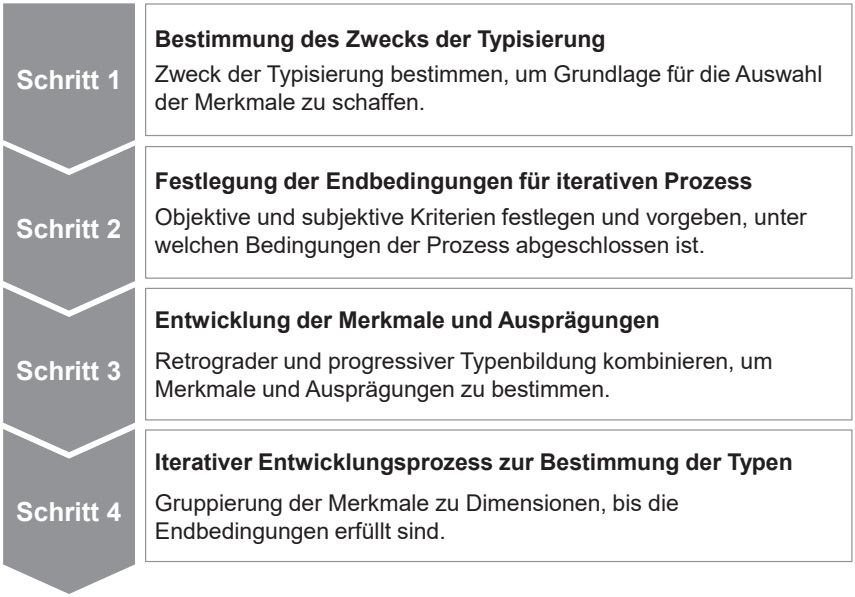

Abbildung 4-5: Prozess der Typenbildung (eigenen Darstellung i. A. a. NICKERSON ET AL. 2013)

Die durch das Vorgehen entwickelten Typen dienen als Grundlage für weitere Analysen und werden im weiteren Verlauf der Arbeit, insbesondere in Kapitel 5, aufgegriffen. Die entwickelten Typen werden mit Erfahrungswissen aus der wissenschaftlichen Literatur und Validierung durch Unternehmensvertreter überprüft (siehe Kapitel 9).

4.2.5 Inhaltsanalyse nach Mayring

Die qualitative Inhaltsanalyse nach MAYRING ist ein systematischer Ansatz zur Textanalyse, der sowohl quantitatives als auch qualitatives Vorgehen integriert. Dieser Ansatz betont die Notwendigkeit eines systematischen, regelgeleiteten Vorgehens bei der Analyse von Kommunikationsmaterial, welches sowohl in Form von Veröffentlichungen als auch in Form von Expert*inneninterviews vorliegen kann. Das systematische und regelgeleitete Vorgehen der qualitativen Inhaltsanalyse ähnelt dem der systematischen Literaturrecherche und macht sie für die Verwendung in einer Dissertation geeignet, da sie so Nachvollziehbarkeit und Überprüfbarkeit gewährleistet.

Bei der qualitativen Inhaltsanalyse wird zwischen der Zusammenfassung, der Explikation und der Strukturierung unterschieden. Das Ziel der Zusammenfassung ist es, das Material so zu reduzieren, dass die wesentlichen Inhalte erhalten bleiben. Bei der Explikation wird zusätzliches Material zu einzelnen fraglichen Textteilen, wie Begriffen oder Sätzen, herangetragen. Bei der Strukturierung geht es darum, bestimmte Aspekte aus dem Material unter vorher festgelegten Ordnungskriterien herauszufiltern. In der vorliegenden Arbeit wird die strukturierte Inhaltsanalyse nach Mayring angewendet, da

aus dem zur Verfügung stehenden Material ein strukturiertes Regelwerk geschaffen werden soll, welches sich an IT-Governance-Ansätzen bedient. Die festgelegten Ordnungskriterien der strukturierten Inhaltsanalyse sollen sich hier an einem IT-Governance-Framework orientieren.

In Anlehnung an MAYRING werden in dieser Arbeit die folgenden fünf Schritte durchlaufen. Im ersten Schritt werden die Strukturierungsdimensionen festgelegt. Diese Dimensionen dienen dazu, das Material systematisch zu analysieren und können entweder deduktiv (theoriegeleitet) oder induktiv (aus dem Material heraus entwickelt) gebildet werden. Im zweiten Schritt werden die Strukturierungsdimensionen weiter differenziert, indem sie in einzelne Ausprägungen unterteilt werden. Anschließend werden im dritten Schritt alle Textelemente, die diesen Strukturierungsdimensionen und deren Ausprägungen entsprechen, systematisch aus dem Material extrahiert und verarbeitet. Im vierten Schritt werden die Ergebnisse des iterativen Prozesses strukturiert aufbereitet. Abschließend, im fünften Schritt, werden die Ergebnisse präsentiert. Der Kern dieses Verfahrens besteht darin, die Strukturierungsdimensionen und deren Ausprägungen präzise zu bestimmen. Der gesamte Ablauf des Verfahrens lässt sich in einem Modell abbilden, welches in Abbildung 4-6 dargestellt ist.

Abbildung 4-6: Vorgehensweise der strukturierten Inhaltsanalyse (eigene Darstellung i. A. a. MAYRING 1984)

In dieser Arbeit wird Mayrings strukturierte Inhaltsanalyse für die Entwicklung des zweiten Beschreibungsmodells in Kapitel 6 verwendet. Die Methode ist hier besonders geeignet, da sich das Modell an den Ordnungskriterien eines IT-Governance Modells bedient und diese auf Basis einer Literaturrecherche gefundenen Quellen für Low-Code neu strukturiert.

4.3 Konkretisierung der Vorgehensweise

In den vorherigen Abschnitten wurden die Ziele dieser Arbeit festgelegt, die Begrifflichkeiten erläutert und die Arbeit eingegrenzt, sowie die Methodik detailliert erörtert, die zur Zielerreichung angewandt wird. Dieser Teil konkretisiert den Forschungsansatz und fasst diesen zusammen.

Die wissenschaftliche Herangehensweise dieser Arbeit erfordert das Erfüllen sowohl formaler als auch inhaltlicher Kriterien, die in Kapitel 4.1.1 und 4.1.2 erläutert wurden. Besonders hervorzuheben sind hier die formal-konzeptionellen Anforderungen an die Reliabilität, Validität und Utilität der entwickelten Modelle, orientiert an der anwendungsorientierten Forschung gemäß den Methoden von ULRICH.

Die Anwendung der Systemtheorie nach HABERFELLNER ET AL., die den Aufbau und die Abgrenzung von Systemen und Subsystemen erläutert, bildet die Grundlage für die inhaltliche Ausrichtung der Forschungsmodelle. Die beiden Beschreibungsmodelle orientieren sich am sozio-technischen Modell nach LEAVITT, welches zur Analyse von organisatorischen Veränderungen eingesetzt wird und sich in die vier Bereiche Aufgabe, Technologie, Akteur und Struktur aufteilen lässt. Das erste Beschreibungsmodell dieser Arbeit bezieht sich auf die Aufgabe, die Technologie und den Akteur, anhand derer drei verschiedene Low-Code-Anwendungsfalltypen abgeleitet werden (siehe Kapitel 5). Das zweite Beschreibungsmodell bezieht sich auf die Struktur und bildet Low-Code-Regeln (siehe Kapitel 6). Daraufhin werden die Ergebnisse der Teilmodelle verknüpft, indem die Wirkbeziehungen zwischen Low-Code-Anwendungsfalltypen und Regeln erklärt werden (siehe Kapitel 7). Anschließend wird ein strukturiertes Regelwerk für den Einsatz von Low-Code-Anwendung in produzierenden Unternehmen gestaltet (siehe Kapitel 8). Insgesamt werden so vier Teilmodelle entwickelt, zwei davon beschreibend, ein Erklärungsmodell und ein Gestaltungsmodell (siehe Abbildung 4-7). Die Ergebnisse werden anschließend in Kapitel 9 evaluiert.

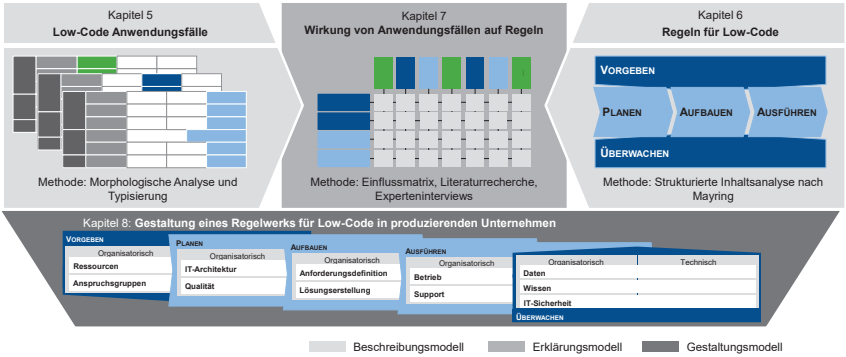

Abbildung 4-7: Übersicht der vier Teilmodelle (eigene Darstellung)

Das Beschreibungsmodell für Low-Code-Anwendungsfälle basiert neben der Untersuchung zu existierenden Low-Code-Anwendungsfällen und -Plattformen in der Praxis

(siehe Kapitel 3.1) und den bestehenden Informationssystem-Architekturen in der Literatur (siehe Kapitel 3.2.2) auf Fallstudien und Erfahrungen der Autorin. Dies entspricht den Ansätzen der Aktionsforschung, bei dem Wissenschaftler*innen und die von ihnen entwickelten Modelle selbst Teil der praktischen Anwendung sind (s. McNIFF U. WHITEHEAD 2006, S. 9). Zuerst werden Merkmale von Low-Code-Anwendungsfällen strukturiert nach den drei Bereichen Aufgabe, Technologie und Akteur mithilfe der morphologischen Analyse (siehe Kapitel 4.2.3) erarbeitet (siehe Kapitel 5.1) und die Merkmalausprägungen abgeleitet (siehe Kapitel 5.2). Die Ableitung der Anwendungsfalltypen (siehe Kapitel 5.3) erfolgt dann auf Basis der Typisierung (siehe Kapitel 4.2.4). Die zentralen Inhalte dieses Modells wurden von der Autorin in FRINGS U. SCHUH 2023 veröffentlicht.

Auch das Beschreibungsmodell der Regeln für Low-Code basiert neben bestehenden IT-Governance Modellen in der Literatur (siehe Kapitel 3.2.3) und einer strukturierten Literaturrecherche (siehe Kapitel 6.1.2) auf Erfahrungen der Autorin. Das Modell stützt sich auf die Struktur des COBIT-Kernmodells und wird mit dem Fokus auf die Implementierung und den Betrieb von Low-Code-Anwendungen adaptiert. Die Low-Code-Regeln werden auf Basis der strukturierten Inhaltsanalyse nach MAYRING identifiziert (siehe Kapitel 6.1) und anschließend mithilfe von Literaturquellen und sachlogischen Ableitungen anhand der Erfahrung der Autorin spezifiziert (siehe Kapitel 6.2).

Das Erklärungsmodell in Kapitel 7 integriert die beiden vorgenannten Modelle und erklärt die Zusammenhänge mit einem Ziel-Zustand, der in diesem Kapitel vorgestellt wird. Die Modellvervollständigung endet mit der Gestaltung des Regelwerks (siehe Kapitel 8). Dieses beschreibt ein Vorgehen für die Anwendung der vorgenannten Modelle in der industriellen Praxis. Die Validierung der Modelle erfolgt in Kapitel 9. Dabei werden die entwickelten Modelle exemplarisch eingesetzt und so verifiziert.

5 Beschreibung der Low-Code-Anwendungsfälle

Das nachfolgende Kapitel dient der Beantwortung der ersten untergeordneten Forschungsfrage:

Wie können Low-Code-Anwendungsfälle in produzierenden Unternehmen beschrieben werden?

Der Fokus liegt darauf, die Anwendungsfälle von Low-Code in produzierenden Unternehmen zu strukturieren und in Form von Typen abzubilden. Zunächst wird der Zweck der Typisierung der Low-Code-Anwendungsfälle bestimmt (siehe Kapitel 5.1.1), um anschließend mithilfe von festgelegten Endbedingungen die detaillierenden Merkmale der Typologie iterativ zu entwickeln (siehe Kapitel 5.1.2). Darauf aufbauend werden die spezifischen Ausprägungen der detaillierenden Merkmale abgeleitet (siehe Kapitel 5.2). Basierend auf den definierten Merkmalen und deren Ausprägungen erfolgt die Herleitung und Vorstellung konsistenter Typen von Low-Code-Anwendungsfällen in produzierenden Unternehmen (siehe Kapitel 5.3) und deren Zusammenfassung (siehe Kapitel 5.4).

Diese erstellte Typologie ermöglicht es, die unterschiedlichen Typen von Low-Code-Anwendungsfällen klar darzustellen. Der gewählte Ansatz zur Entwicklung der Typologie orientiert sich dabei an NICKERSON ET AL. (siehe Kapitel 4.2.4) und ist in Abbildung 5-1 visualisiert.

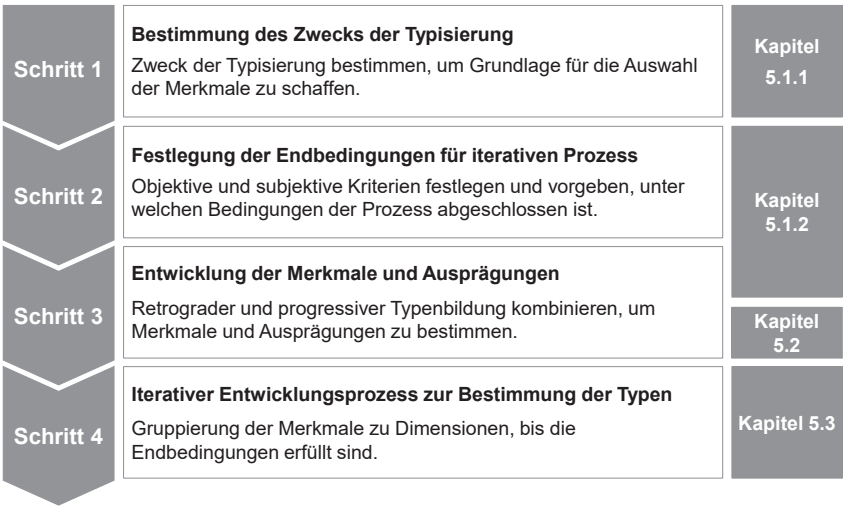

Abbildung 5-1: Vorgehensweise der Typisierung (eigene Darstellung i. A. a. NICKERSON ET AL. 2013)

5.1 Merkmale der Low-Code-Anwendungsfälle

Im Folgenden soll zunächst der Zweck der Typisierung für Low-Code-Anwendungsfälle näher beschrieben werden. Anschließend folgt die Beschreibung der Merkmale.

5.1.1 Zweck der Typisierung

Für die Festlegung der Merkmale ist zunächst zu entscheiden, was der konkrete Zweck der Typisierung ist (siehe Kapitel 4.2.4). Dazu wird im Folgenden ein Bild eines Anwenders der Forschungsergebnisse gezeichnet, um darauf basierend den Mehrwert abzuleiten.

Die Ausgangssituation beschreibt ein Unternehmen, das beabsichtigt, Low-Code-Anwendungsfälle im Produktionsumfeld umzusetzen. Das höhere Abstraktionsniveau und die visuelle Entwicklungsumgebung von Low-Code-Plattformen ermöglichen es, neben IT-Bereichen auch weitere Mitarbeiter*innen aus den Produktions- und Fachbereichen in den Entwicklungsprozess von Low-Code-Anwendungen zu integrieren. Für die Mitarbeiter*innen ist jedoch noch nicht geklärt, wer welche Anwendungen mit Low-Code entwickeln soll. Außerdem befinden sie sich entweder in der Entscheidungsfindung, welche Low-Code-Plattform für die Umsetzung der anvisierten Anwendungen in Betracht gezogen werden soll, oder haben diese Entscheidung gerade getroffen. Der aktuelle Wissensstand des Unternehmens darüber, welche Art von Anwendung mit welcher Low-Code-Plattform umgesetzt werden kann, ist gering. Ebenso ist noch unklar, welche Fähigkeiten notwendig sind, um spezifische Low-Code-Anwendungen zu entwickeln.

Der Mehrwert der Typisierung wird deutlich, da bisher keine Erfahrungen mit Low-Code-Anwendungsfällen vorliegen. Die Typisierung soll das Verständnis für den möglichen Umfang von Low-Code-Anwendungen verbessern und Transparenz über die notwendigen Fähigkeiten der Low-Code-Entwickler*innen sowie über die Funktionalitäten der Low-Code-Plattform schaffen. Sie soll dabei helfen, zu bestimmen, welche Low-Code-Anwendungen mit welchen Low-Code-Funktionalitäten und welchen Programmierfähigkeiten realisiert werden können. So bietet die Typisierung eine umfassende Perspektive auf potenzielle Low-Code-Anwendungsfälle.

5.1.2 Detaillierte Merkmale

Nachdem im vorherigen Kapitel der Zweck der Typisierung bestimmt wurde, liegt nun der Fokus auf den detaillierenden Merkmalen für Low-Code-Anwendungsfälle.

Zur systematischen Kategorisierung dient das sozio-technische Rahmenkonzept von LEAVITT (siehe Abbildung 5-2, LEAVITT 1962). Dieses erlaubt eine effektive Strukturierung in die drei Dimensionen – *Aufgabe, Technologie* und *Akteur* – und verschafft somit eine umfassende Perspektive auf die Low-Code-Anwendungsfälle in produzierenden Unternehmen.

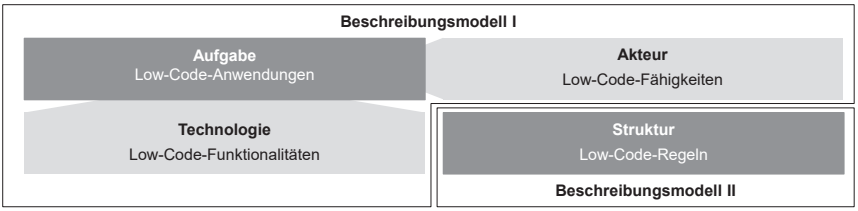

Abbildung 5-2: Sozio-technischer Rahmen für Low-Code (eigene Darstellung i. A. a. LEAVITT 1962)

Die Unterscheidung zwischen Aufgabe, Technologie und Akteur hat das Ziel, sowohl eine technische als auch eine organisatorische Perspektive für die Umsetzung von Low-Code-Anwendungsfällen zu berücksichtigen. Die organisatorische Perspektive beschreibt die Rahmenbedingungen innerhalb des Unternehmens in denen die Low-Code-Anwendung zum Einsatz kommen. Die technische Perspektive beschreibt relevante Elemente einer Low-Code-Plattform, die Auswirkungen auf die Funktionsweise der Low-Code-Anwendung hat. Abgeleitet von den drei Bereichen *Aufgabe, Technologie* und *Akteur* werden anknüpfend die Merkmale und deren Ausprägungen entwickelt.

Der Entwicklungsprozess der Typologie soll bei der vorliegenden Arbeit so lange durchgeführt werden, bis alle von *NICKERSON* ET AL. empfohlenen *subjektiven Endbedingungen* erfüllt sind. In der Tabelle 16 sind diese Forderungen aufgelistet und durch jeweilige Fragestellungen konkretisiert (s. NICKERSON ET AL. 2013, S. 344).

Tabelle 16: Übersicht der subjektiven Endbedingungen des Iterationsprozesses (i. A. a. NICKERSON ET AL. 2013, S. 344)

Bedingung	Fragestellung
prägnant	Erlaubt die Anzahl der Merkmale, dass der morphologische Kasten sinnvoll erscheint, ohne schwerfällig oder überwältigend zu wirken?
robust	Ermöglichen die Merkmale und Ausprägungen eine ausreichende Differenzierung zwischen den untersuchten Objekten?
umfassend	Können alle Objekte oder eine (zufällige) Stichprobe an Objekten innerhalb des Untersuchungs-bereichs beschrieben werden? Sind alle Merkmale der relevanten Objekte identifiziert?
erweiterbar	Kann ein neues Merkmal oder eine neue Ausprägung eines bestehenden Merkmals leicht hinzugefügt werden?
erklärend	Was sagen die Merkmale und Ausprägungen über das Untersuchungsobjekt aus?

In Bezug auf die *objektiven Endbedingungen* werden die folgenden drei Kriterien gewählt:

(1) Jedes Merkmal ist einzigartig und nicht wiederholt
(2) Keine neuen Merkmale oder Ausprägungen wurden in der letzten Iteration hinzugefügt
(3) Jede Ausprägung ist innerhalb ihres Merkmals einzigartig

NICKERSON ET AL. haben in ihrer Arbeit eine Reihe an objektiven Endbedingungen mit deren individueller Relevanz erläutert. Zu der unter (1) festgelegten Bedingung begründen sie, dass sich im Falle einer Ambivalenz zwischen Merkmalen ungewollte Redundanzen bilden, die für eine aussagekräftige Typologie unbedingt eliminiert werden müssen. Zudem ist es entscheidend, dass im Falle eines Zusammenfügens oder Aufteilens von Ausprägungen eines Merkmals die Auswirkungen dieser Änderung auf die Typologie untersucht werden. Der Endzustand dieser Untersuchung ist erst erreicht, wenn die in (2) gelistete Bedingung erfüllt ist. Für den Fall, dass ein Merkmal keine einzigartige Ausprägung des zu untersuchenden Objektes aufweist, muss diese Ausprägung entweder entfernt, oder ein Objekt im Untersuchungsbereich mit einer eindeutigeren Merkmalausprägung identifiziert werden. Um dies sicherzustellen, haben NICKERSON ET AL. die dritte Endbedingung (3) vorgeschlagen. (s. NICKERSON ET AL. 2013, S. 344)

Die oben aufgeführten, objektiven Endbedingungen entsprechen denen, die bereits in der Vergangenheit für erfolgreiche Entwicklungsprozesse von Typologien im Bereich der Informationstechnologien genutzt wurden (s. PÜSCHEL ET AL. 2016, S. 5; ARNOLD ET AL. 2021, S. 408). Je nach Auswahl der Endbedingungen kann es jedoch dazu kommen, dass unterschiedliche morphologische Kästen und damit verschiedene Typologien erzeugt werden. Dies ist trotz alledem konsistent mit dem iterativen Prinzip der Designwissenschaft bei Informationssystemen, nicht unbedingt optimale, dafür aber sinnvolle Lösungen für reale Geschäftsprobleme zu entwickeln (s. HEVNER ET AL. 2004, S. 88).

Nach Abschluss der letzten Iteration wird der morphologische Kasten final einer tiefgehenden Untersuchung zur Erfüllung der restlichen Endbedingungen unterzogen. Hierdurch wurden besonders Begrifflichkeiten präzisiert und etwaige Dopplungen in den Ausprägungen entfernt. Abbildung 5-3 zeigt die ermittelten detaillierenden Merkmale, die im folgenden Kapitel inklusive ihrer Ausprägungen vorgestellt werden.

Aufgabe			Technologie		Akteur
Geschäfts-kritikalität	Nutzung	Lebenszyklus	Anpassungs-möglichkeit	Schnittstellen	Programmier-kenntnisse

Abbildung 5-3: Detaillierende Merkmale der Low-Code-Anwendungsfälle (eigene Darstellung)

5.2 Merkmalausprägungen der Low-Code-Anwendungsfälle

Die identifizierten sechs Merkmale zusammen mit ihren siebzehn Ausprägungen bieten wertvolle Einblicke in die wesentlichen Aspekte von Low-Code-Anwendungsfällen in produzierenden Unternehmen. Die Kategorisierung der Ausprägungen erfolgt durch klassifizierende und graduierbare Merkmale, wobei bewusst von der Verwendung quantifizierbarer Variablen abgesehen wird, um eine Scheingenauigkeit zu vermeiden (vgl. KRÄMER 2014, S. 84-85). Dieser Ansatz trägt der subjektiven Natur qualitativer Forschung Rechnung und vereinfacht gleichzeitig die Komplexität, um ein praxisgerechtes Konzept für den Unternehmenseinsatz zu entwickeln (vgl. SIEGERS 2016, S. 149).

In den nachfolgenden Abschnitten werden die detaillierenden Merkmale präsentiert und deren Ausprägungen, basierend auf umfassenden Literaturstudien, näher beschrieben.

5.2.1 Aufgabe

Geschäftskritikalität

Geschäftskritikalität beschreibt das Ausmaß der Bedeutung einer Anwendung und ihrer Daten für die Wertschöpfung eines Unternehmens. Die Ausprägungen dieses Merkmals orientieren sich an den drei Kategorien aus dem Bereich des Datenmanagements (siehe Kapitel 2.2.1): *nicht kritisch, kritisch und geschäftskritisch* (s. MAHANTI 2021, S. 34) (siehe Abbildung 5-4). Der Ausfall nicht kritischer Anwendungen oder der Verlust nicht kritischer Daten wird als vernachlässigbar für die Gesamtfunktionen eines Unternehmens betrachtet. Da sie keinen direkten Einfluss auf wertschöpfende Prozesse haben, führt deren Ausfall oder Datenverlust nur zu geringfügigen Effizienzverlusten für das Unternehmen. Andererseits kann der Ausfall kritischer Anwendungen oder der Verlust kritischer Daten den regulären Betrieb und die Abläufe im Unternehmen stören. Obwohl sie keine direkten Auswirkungen auf wertschöpfende Prozesse haben, führen ihr Ausfall und Verlust zu spürbaren Effizienzverlusten und kann Störungen in wertschöpfenden Aktivitäten verursachen. Darüber hinaus birgt der Ausfall geschäftskritischer Anwendungen oder der Verlust geschäftskritischer Daten ein erhebliches Risiko, da die Daten eng mit wertschöpfenden Prozessen verknüpft sind und direkten Einfluss auf diese haben. Der Ausfall solcher Anwendungen und der Verlust der Daten würden die Wertschöpfung erheblich beeinträchtigen und möglicherweise zu schwerwiegenden Konsequenzen, wie der Stilllegung einer wichtigen Einrichtung, führen (s. MAHANTI 2021, S. 34).

Geschäftskritikalität	Unkritisch	Kritisch	Geschäftskritisch

Abbildung 5-4: Ausprägungen des Merkmals „Geschäftskritikalität“ (eigene Darstellung)

Die Geschäftskritikalität einer Anwendung und ihrer Daten bestimmt die Bedeutung der Anwendung hinsichtlich Zuverlässigkeit und Sicherheit. Der Ausfall von Systemen,

die als geschäftskritisch eingestuft werden, haben einen großen Einfluss auf Geschäftsabläufe, Wirtschaftlichkeit und Wettbewerbsvorteile (s. SCHWARTZEL 2012). Daher ist ein höheres Maß an Zuverlässigkeit und Sicherheit erforderlich, um das Risiko von Ausfällen oder Verstößen zu minimieren, die zu erheblichen Schäden oder Unterbrechungen des Geschäftsbetriebs führen könnten.

Einsatz im Unternehmen

Die Bestimmung der Zielgruppe ist von entscheidender Bedeutung für die Spezifikation der Anforderungen an die Anwendung (siehe Kapitel 2.1.3). Diese Definition legt fest, welche Bereiche innerhalb der Organisation von der Anwendungsentwicklung betroffen sind (s. POHL U. RUPP 2015). Für die IT-Architektur ist es ebenso entscheidend, die Integrationsreichweite zu bestimmen, die festlegt, ob die Integration innerhalb eines Bereichs, innerhalb eines Standorts oder standortübergreifend stattfindet. Diese Bestimmung ist ausschlaggebend für eine nachhaltige Integration der Low-Code-Plattform in die IT-Architektur (siehe Kapitel 2.2.3). Der Einsatz im Unternehmen lässt sich daher in abteilungsspezifische Einflüsse sowie in bereichs- und standortübergreifende Einflüsse unterteilen (siehe Abbildung 5-5). Abteilungsspezifische Anwendungen erfüllen die speziellen Bedürfnisse und Arbeitsprozesse einer Abteilung und sind möglicherweise für andere Bereiche irrelevant. Andere Anwendungen können eine breitere Anwendung finden und in mehreren Abteilungen innerhalb der Organisation von Bedeutung sein. Sie dienen dazu, den Informationsaustausch zwischen verschiedenen Teams zu fördern und werden daher in eine größere Anzahl von anderen Anwendungen integriert. In größeren Organisationen mit mehreren Standorten können einige Anwendungen über mehrere Standorte hinweg genutzt werden und verfügen über diverse Anwendungsintegrationen, die eine nahtlose Kommunikation und Koordination über die geografische Ausdehnung des Unternehmens hinweg ermöglichen.

Nutzung	Abteilungsintern	Abteilungsübergreifend	Standortübergreifend

Abbildung 5-5: Ausprägungen des Merkmals „Einsatz im Unternehmen“ (eigene Darstellung)

Die Nutzung spielt eine entscheidende Rolle bei Entscheidungen zur Anwendungsentwicklung und -bereitstellung. Anwendungen mit einer breiteren Anwenderbasis können eine gründlichere Kontrolle, Prüfung und weitere Sicherheitsmaßnahmen erfordern, um ihre Leistungseffizienz, Benutzerfreundlichkeit, funktionale Eignung und Kompatibilität mit anderen Anwendungen zu gewährleisten.

Lebensdauer

Die Lebensdauer einer Anwendung bezieht sich auf die Dauer, für die die Anwendung innerhalb des Unternehmens verwendet wird. Im Pace-Layered Application Modell (siehe Kapitel 3.2.2) haben Anwendungen mit schnell wechselnden Anforderungen in der Regel eine kürzere Lebensdauer im Vergleich zu Anwendungen, die Kerngeschäftsprozesse unterstützen und eine stabilere, langfristige Nutzung haben (s. MESAGLIO U. HOTLE 2016). Die Lebensdauer von Anwendungen wird in drei Hauptdauern

kategorisiert: wenige Jahre (1-2 Jahre), mehrere Jahre (3-5 Jahre) und viele Jahre (5-10 Jahre) (siehe Abbildung 5-6). Anwendungen mit einer Lebensdauer von wenigen Jahren müssen aufgrund der schnellen technologischen Entwicklung und sich schnell ändernden Geschäftsanforderungen regelmäßig aktualisiert oder ersetzt werden. Für solche Anwendungen ist Flexibilität entscheidend, da sie häufig neue Funktionen hinzufügen oder bestehende anpassen müssen, um wettbewerbsfähig zu bleiben. Anwendungen mit einer Lebensdauer von mehreren Jahren erfordern ein ausgewogenes Maß an Flexibilität und Stabilität. Sie müssen in der Lage sein, sich Veränderungen anzupassen, aber nicht so häufig wie Anwendungen mit einer kurzen Lebensdauer. Für diese Kategorie ist es wichtig, eine robuste Architektur zu haben, die sowohl Erweiterbarkeit als auch Wartbarkeit unterstützt. Anwendungen mit einer Lebensdauer von fünf bis zehn Jahren unterstützen Kerngeschäftsprozesse und erfordern ein hohes Maß an Wartbarkeit und Zuverlässigkeit. Solche Anwendungen müssen über Jahre hinweg stabil laufen können, ohne dass häufige große Änderungen erforderlich sind. Die Entwicklung solcher Systeme erfordert eine gründliche Planung und die Berücksichtigung zukünftiger Anforderungen bereits in der Entwurfsphase. Es ist von entscheidender Bedeutung, eine solide Grundlage zu schaffen, die spätere Anpassungen und Erweiterungen erleichtert, ohne die Gesamtstabilität zu beeinträchtigen.

Lebenszyklus	Wenige Jahre (1-2)	Mehrere Jahre (3-5)	Viele Jahre (5-10)

Abbildung 5-6: Ausprägungen des Merkmals „Lebensdauer“ (eigene Darstellung)

Die Lebensdauer einer Anwendung beeinflusst direkt die Prioritäten in ihrer Entwicklung und Wartung. Während Anwendungen mit kurzer Lebensdauer Agilität und schnelle Anpassungsfähigkeit erfordern, benötigen langlebige Anwendungen eine sorgfältige Planung und eine robuste Architektur, um langfristige Stabilität und Wartbarkeit zu gewährleisten.

5.2.2 Technologie

Anpassungsfähigkeit

Die Charakteristik der Anpassungsfähigkeit im Low-Code-Entwicklungsprozess bezieht sich auf das Ausmaß, in dem Benutzer*innen Anpassungen oder Änderungen an den Bausteinen und Modellen innerhalb der Entwicklungsumgebung vornehmen können. Dabei werden zwei Ausprägungen unterschieden: Anpassung durch Low-Code und Anpassung im Source Code (s. Bock u. Frank 2021) (siehe Abbildung 5-7). Bei Anpassung durch Low-Code sind Benutzer*innen darauf beschränkt, Änderungen nur am Standardsystem der Low-Code-Plattform vorzunehmen. Weitere Modifikationen oder Anpassungen müssen beim Anbieter der Low-Code-Plattform beantragt werden. In diesem Szenario haben Benutzer*innen nur begrenzten Einfluss auf den Anpassungsprozess und Änderungen sind in der Regel auf vordefinierte Optionen des Anbieters beschränkt (s. Jednaszewski 2024). Auf der anderen Seite ermöglicht eine Anpassung durch den Source Code es Entwickelnden, Artefakte innerhalb der Entwicklungsumgebung anzupassen. Dieses Maß an Anpassungsfähigkeit bietet größere

Flexibilität und ermöglicht Entwicklern, die Low-Code-Anwendung auf ihre spezifischen Bedürfnisse und Anforderungen zuzuschneiden.

Anpassungsmöglichkeit	Low-Code	Source Code

Abbildung 5-7: Ausprägungen des Merkmals „Anpassungsfähigkeit“ (eigene Darstellung)

Höhere Anpassungsfähigkeit führt auch zu einer größeren Komplexität bei der Entwicklung und Verwaltung von Low-Code-Anwendung. Dies liegt daran, dass eine systematische Verwaltung von Software-Artefakten essenziell ist, um die technische Komplexität in der Softwareentwicklung zu beherrschen (s. Broy u. Kuhrmann 2021).

Schnittstellen

Die Ausprägungen des Merkmals der Schnittstellen in der Low-Code-Entwicklung bezieht sich auf die Fähigkeit der Technologie, die Integration neuer Datenquellen und Anwendungen in die Low-Code-Plattform als auch die Integration bestehender Low-Code-Anwendung mit neuen Diensten und Daten zu unterstützen (s. Oteyo et al. 2021). Bei einigen Low-Code-Plattformen sind die Integrationsfähigkeiten auf standardisierte Schnittstellen bestimmter Anbieter limitiert, was die Benutzer*innen dazu zwingt, ausschließlich diese vorgegebenen Schnittstellen für die Anbindung an externe Datenquellen oder Dienste zu nutzen (s. Bock u. Frank 2021). Im Gegensatz dazu ermöglichen es andere Plattformen, individuelle Schnittstellen zu verwenden. Diese Flexibilität erlaubt es den Nutzern, eine breite Palette von Software-Diensten und Daten zu integrieren (s. Bock u. Frank 2021). Im Rahmen dieser Arbeit wird zwischen standardisierten, konfigurierbaren und individuellen Schnittstellen unterschieden (siehe Abbildung 5-8, Kapitel 2.1.3). Standardisierte Schnittstellen beschränken sich auf ein vorab definiertes Austauschformat zwischen der Low-Code-Plattform und dem jeweiligen System. Konfigurierbare Schnittstellen erlauben eine Anpassung der Austauschformat-Parameter zwischen Low-Code-Plattform und Anwendung. Individuelle Schnittstellen bieten die Möglichkeit, die anzubindende Anwendung sowie das gesamte Datenaustauschformat frei zu wählen.

Schnittstellen	Standardisiert	Konfigurierbar	Individuell

Abbildung 5-8: Ausprägungen des Merkmals „Schnittstellen“ (eigene Darstellung)

Vielseitige Schnittstellen in der Low-Code-Entwicklung bieten Entwickelnden die Möglichkeit, komplexe Anwendungen zu erstellen, die eine nahtlose Interaktion mit unterschiedlichen Datenquellen und Diensten ermöglichen. Diese Flexibilität ist entscheidend für die Entwicklung von Anwendungen, die auf ein breites Spektrum an Funktionen und Daten zugreifen müssen. Auf der anderen Seite sind Low-Code-Plattformen mit eingeschränkten Integrationsmöglichkeiten besonders geeignet für die Erstellung einfacherer Anwendungen. Sie bieten den Vorteil, dass durch die Begrenzung der externen Verbindungen Sicherheitsrisiken minimiert werden können (s. Heuer et al. 2022).

5.2.3 Akteur

Programmierfähigkeiten

Die Charakteristik der Programmierfähigkeiten beschreibt das Kompetenzniveau, das Mitarbeiter*innen aufweisen. Es werden drei Ausprägungen unterschieden: keine, geringe und professionelle Programmierfähigkeiten (siehe Abbildung 5-9), was mit der Differenzierung zwischen „Citizen Developers" und „Professional Developers" übereinstimmt (siehe Kapitel 2.1.4). GARTNER unterscheidet innerhalb der Gruppe der *Citizen Developers* zwischen „Power Users“ und „End Users“. End Users sind Personen ohne Programmierfähigkeiten, die Anwendungen primär zur Steigerung ihrer persönlichen Effizienz am Arbeitsplatz und zur Optimierung individueller Arbeitsabläufe erstellen. Power Users hingegen sind versierte Mitarbeitende mit grundlegenden Programmierfähigkeiten, die Low-Code-Plattformen nutzen, um anspruchsvollere Softwareanwendungen für eine breitere Nutzerbasis zu entwickeln (s. WONG 2019). Professionelle Entwickelnde zeichnen sich durch eine formale Ausbildung oder signifikante Erfahrung in der Softwareentwicklung aus und arbeiten traditionell mit textbasierten Programmiersprachen wie C++, Java oder Python (s. HIRZEL 2022). GARTNER definiert professionelle Entwickelnde als Vollzeitkräfte, was ihre spezialisierte Ausrichtung auf Entwicklungsprojekte unterstreicht.

Programmierfähigkeiten	Keine	Wenig	Erfahren

Abbildung 5-9: Ausprägungen des Merkmals „Programmierfähigkeiten“ (eigene Darstellung)

Das Niveau der Programmierfähigkeiten eines Low-Code-Entwickelnden spielt eine entscheidende Rolle für die Komplexität und den Funktionsumfang der Anwendungen, die sie entwickeln können. Entwickelnde mit geringen oder keinen Programmierfähigkeiten sind in der Regel auf die Erstellung einfacher und grundlegender Anwendungen limitiert. Auf der anderen Seite können professionelle Entwickelnde, die über tiefgreifende Programmierfähigkeiten verfügen, die Grenzen der Low-Code-Plattformen überschreiten. Sie sind in der Lage, anspruchsvolle und komplexe Anwendungen zu erstellen, die genau auf die spezifischen Anforderungen und Funktionalitäten abgestimmt sind, die von den Benutzern benötigt werden.

5.2.4 Zusammenfassung

Nach der detaillierten Beschreibung der Merkmale und ihren Ausprägungen in den beiden vorherigen Kapiteln zeigt Abbildung 5-10 nun die Integration dieser Merkmale in eine Typologie.

Aufgabe	Geschäftskritikalität	Unkritisch	Kritisch	Geschäftskritisch
	Nutzung	Abteilungsintern	Abteilungsübergreifend	Standortübergreifend
	Lebenszyklus	Wenige Jahre (1-2)	Mehrere Jahre (3-5)	Viele Jahre (5-10)
Technologie	Anpassungsmöglichkeit	Low-Code		Source Code
	Schnittstellen	Standardisiert	Konfigurierbar	Individuell
Akteur	Programmierfähigkeiten	Keine	Wenig	Erfahren

Abbildung 5-10: Merkmale und Ausprägungen von Low-Code-Anwendungsfällen (eigene Darstellung)

Die Darstellung entspricht den Ansprüchen der Literatur für eine Morphologie (s. WELTER 2006, S. 115-116; NICKERSON ET AL. 2013, S. 342) und sortiert die Ausprägungen von links nach rechts aufsteigend ordinalskaliert. Dies bedeutet, dass Komplexität und Umfang der Funktionalitäten steigen, je weiter diese auf der rechten Seite des Kastens eingeordnet sind. Die detaillierten Merkmale und ihre abgeleiteten Variationen ermöglichen die Identifizierung verschiedener Typen von Low-Code-Anwendungsfällen in produzierenden Unternehmen.

5.3 Typen der der Low-Code-Anwendungsfälle

Zur Unterstützung von produzierenden Unternehmen bei der Implementierung und dem Betrieb von Low-Code-Anwendung soll als Grundlage eine Typisierung dienen. Nachdem im vorherigen Kapitel die morphologische Analyse abgeschlossen und Charakteristiken der Low-Code-Anwendungsfälle identifiziert wurden, werden in diesem Kapitel nun repräsentative Typen vorgestellt. Am Ende der Vorstellung jedes Typen wird ein konkreter Anwendungsfall des Anwendungsfalltypen innerhalb eines produzierenden Unternehmens erläutert.

Die konsistenten Typen von Low-Code-Anwendungsfällen werden anhand der beschriebenen Merkmale und ihrer Ausprägungen identifiziert. Hierzu werden die verschiedenen Ausprägungen der Merkmale gemäß der Konfigurationstheorie und dem Fit-Konzept zu zusammenhängenden Typen kombiniert (vgl. SIEGERS 2016, S. 161-162). Es wird erneut hervorgehoben, dass die identifizierten idealen Typen in der realen Welt tatsächlich vorkommen können. Die Anwendungsfälle im Bereich von Low-Code können entweder genau dem idealen Typus entsprechen oder zumindest gewisse Ähnlichkeiten aufweisen (s. SIEGERS 2016, S. 161). Hierbei wird nach der Methodik von WELTER insbesondere geprüft, ob die einzelnen Merkmale und ihre Ausprägungen voneinander abgegrenzt sind und ob möglicherweise Kombinationen von Merkmalen existieren können (s. WELTER 2006, S. 116).

Um die Präzision und Anwendbarkeit der entwickelten Typen zu gewährleisten, wurden die einzelnen Merkmale und ihre Ausprägungen sowohl in Gesprächen mit Expert*innen (siehe Kapitel 5.3.2 bis 5.3.4) als auch durch praktische Überprüfung validiert (siehe Kapitel 9). In den Validierungsgesprächen wurden Anwendungen, die mithilfe von Low-Code entwickelt wurden, gemeinsam mit den Expert*innen analysiert.

Ziel war es, diese Beispiele in die erstellte Typologie einzuordnen und sie mit den identifizierten Idealtypen in dieser Arbeit zu vergleichen.

5.3.1 Herleitung konsistenter Typen von Low-Code-Anwendungsfällen

Basierend auf den in Kapitel 5.2 abgeleiteten Merkmalen lässt sich eine klare Identifikation und Beschreibung konsistenter Low-Code-Anwendungsfalltypen gewährleisten. Die bisherige Literatur bietet keine Typisierung von Low-Code-Anwendungsfällen (siehe Kapitel 3.1.1). Es lassen sich lediglich vereinzelt Ansätze zur Typisierung von Low-Code-Plattformen identifizieren, wobei die zentralen Unterscheidungsmerkmale größtenteils technischer Natur sind (siehe Kapitel 3.1.2).

Basierend auf den in der Literatur identifizierten Merkmalen für Low-Code-Anwendung (siehe Kapitel 3.1) sowie den Erkenntnissen aus den Modellen zu Informationssystem-Architekturen (siehe Kapitel 3.2.2) können verschiedene Typen von Low-Code-Anwendungsfällen abgeleitet werden. Dieser Ansatz stützt sich auf die Vorgehensweise der Typisierung nach WELTER, indem logisch widerspruchsfreie, empirisch verifizierbare und praktisch brauchbare Verknüpfung von Merkmalsausprägungen untersucht werden. Um die Verknüpfungen zwischen den Merkmalsausprägungen zu bestimmen, wird, kombiniert mit dem Vorverständnis der Autorin, auf die in der Literatur vorhandenen Abhängigkeiten zwischen den Merkmalen sowie auf bereits durchgeführte Analysen zurückgegriffen (s. MAYRING 2002, S. 25). Die Verknüpfung unterschiedlicher Merkmalsausprägungen soll auf ihren Zusammenhang, im Folgenden als „Fit“ bezeichnet, hin untersucht werden. Die Sicherstellung der Richtigkeit der nachfolgenden „Fit“-Bewertung wird durch wiederholte Diskussion und Validierung in Expert*innengesprächen sichergestellt (siehe Kapitel 9).

Die Bewertung des „Fits" zwischen zwei Merkmalsausprägungen erfolgte anhand von drei Kombinationskriterien, die im Folgenden kurz erläutert werden:

- **Positiver Zusammenhang (+):** Wenn zwischen den beiden Merkmalsausprägungen eine klare Abhängigkeit und logische Verknüpfung besteht, wird ein starker „Fit" angenommen. Die Kombination dieser Merkmalsausprägungen unterstützt die Erreichung eines konsistenten Idealtyps.
- **Neutral (o):** Es liegt keine starke Abhängigkeit zwischen den beiden Merkmalsausprägungen vor. Dennoch wird der „Fit" nicht als hinderlich für die Erreichung eines konsistenten Idealtyps betrachtet und somit nicht negativ bewertet. Es besteht weder eine fördernde noch hemmende Abhängigkeit zwischen den Merkmalsausprägungen.
- **Negativer Zusammenhang (-):** Wenn die beiden Merkmalsausprägungen einen Widerspruch darstellen, ergibt sich ein negativer „Fit". In diesem Fall führt die Kombination der beiden Ausprägungen nicht zu einem passenden „Fit" und könnte die Bildung eines konsistenten Typs negativ beeinflussen oder sogar verhindern.

Unter Anwendung dieser drei Bewertungsausprägungen wurde der „Fit" zwischen den verschiedenen Merkmalsausprägungen analysiert und in einer visualisierten Darstellung abgebildet, wie in Abbildung 5-11 dargestellt.

		Geschäftskritikalität			Nutzung			Lebensdauer			Anpassungen		Schnittstellen			Programmierfähigkeiten		
		Unkritisch	Kritisch	Geschäftskritisch	Abteilungsintern	Abteilungsübergreifend	Standortübergreifend	Wenige Jahre	Mehrere Jahre	Viele Jahre	Low-Code	Source Code	Standardisiert	Konfigurierbar	Individuell	Keine	Wenig	Erfahren
Geschäftskritikalität	Unkritisch				+	0	0	0	0	0	+	0	0	0	0	+	0	0
	Kritisch				0	+	0	0	0	0	0	0	0	0	0	0	+	0
	Geschäftskritisch				0	0	+	0	0	0	-	+	0	0	0	-	0	+
Nutzung	Abteilungsintern	+	0	0				+	0	0	0	0	0	0	0	+	0	0
	Abteilungsübergreifend	0	+	0				0	+	0	0	0	0	0	0	0	+	0
	Standortübergreifend	0	0	+				0	0	+	0	+	-	0	+	-	0	+
Lebensdauer	Wenige Jahre	0	0	0	+	0	0				0	0	0	0	0	+	0	0
	Mehrere Jahre	0	0	0	0	+	0				0	0	0	0	0	0	+	0
	Viele Jahre	0	0	0	0	0	+				-	+	0	0	0	-	0	+
Anpassungsmöglichkeit	Low-Code	+	0	-	0	0	0	0	0	-			+	+	0	+	+	0
	Source Code	0	0	+	0	0	+	0	0	+			0	0	+	-	0	+
Schnittstellen	Standardisiert	0	0	0	0	0	-	0	0	0	+	0				+	0	0
	Konfigurierbar	0	0	0	0	0	0	0	0	0	+	0				0	+	0
	Individuell	0	0	0	0	0	+	0	0	0	0	+				-	0	+
Programmierfähigkeiten	Keine	+	0	-	+	0	-	+	0	-	+	-	+	0	-			
	Wenig	0	+	0	0	+	0	0	+	0	+	0	0	+	0			
	Erfahren	0	0	+	0	0	+	0	0	+	0	+	0	0	+			

Abbildung 5-11: „Fit“ der Ausprägungen aller detaillierten Merkmale von Low-Code-Anwendungsfällen (eigene Darstellung)

Basierend auf den durchgeführten „Fit“-Bewertungen der Merkmalsausprägungen untereinander lassen sich nun konsistente Typen von Low-Code-Anwendungsfällen ableiten. Hierbei werden Kombinationen von Ausprägungen identifiziert, die idealerweise einen positiven Zusammenhang aufweisen. Der „Fit“ sollte nicht nur zwischen zwei Merkmalsausprägungen bestehen, sondern am besten zwischen allen Ausprägungen eines Typs. Ausgehend von der Geschäftskritikalität werden alle Kombinationen schrittweise überprüft. Anschließend können die anderen Merkmale dahingehend eingestuft werden, ob sie die Bildung eines konsistenten Typs fördern oder behindern. Wenn bei einer Kombination von Ausprägungen weder ein positiver noch ein negativer Zusammenhang festgestellt wird, wird auf die neutrale Bewertungsmöglichkeit zurückgegriffen. Negative Zusammenhänge (-) zwischen zwei Merkmalsausprägungen sollten bei der Bildung konsistenter Typen vermieden werden, da ein negativer Zusammenhang die Bildung eines Idealtyps verhindert. Somit sollten die

Merkmalsausprägungen bei der Bildung eines Typs mindestens einen neutralen (o) und idealerweise einen positiven (+) Zusammenhang untereinander aufweisen.

Für Low-Code-Anwendungsfälle wurden drei verschiedene Ausprägungskombinationen identifiziert, bei denen ein Gesamt-„Fit“ vorliegt. Diese werden in Abbildung 5-12 mithilfe verschiedener Farbtöne visualisiert.

		Geschäfts-kritikalität			Nutzung			Lebens-dauer			Anpas-sungen		Schnitt-stellen			Programmier-fähigkeiten		
		Unkritisch	Kritisch	Geschäftskritisch	Abteilungsintern	Abteilungsübergreifend	Standortübergreifend	Wenige Jahre	Mehrere Jahre	Viele Jahre	Low-Code	Source Code	Standardisiert	Konfigurierbar	Individuell	Keine	Wenig	Erfahren
Geschäfts-kritikalität	Unkritisch				+	0	0	0	0	0	+	0	0	0	0	+	0	0
	Kritisch				0	+	0	0	0	0	0	0	0	0	0	0	+	0
	Geschäftskritisch				0	0	+	0	0	0	-	+	0	0	0	-	0	+
Nutzung	Abteilungsintern	+	0	0				+	0	0	0	0	0	0	0	+	0	0
	Abteilungsübergreifend	0	+	0				0	+	0	0	0	0	0	0	0	+	0
	Standortübergreifend	0	0	+				0	0	+	0	+	-	0	+	-	0	+
Lebensdauer	Kurz	0	0	0	+	0	0				0	0	0	0	0	+	0	0
	Mittel	0	0	0	0	+	0				0	0	0	0	0	0	+	0
	Lang	0	0	0	0	0	+				-	+	0	0	0	-	0	+
Anpassungs-möglichkeit	Nur Low-Code	+	0	-	0	0	0	0	0	-			+	+	0	+	+	0
	Auch Source Code	0	0	+	0	0	+	0	0	+			0	0	+	-	0	+
Schnittstellen	Standardisiert	0	0	0	0	0	-	0	0	0	+	0				+	0	0
	Konfigurierbar	0	0	0	0	0	0	0	0	0	+	0				0	+	0
	Individuell	0	0	0	0	0	+	0	0	0	0	+				-	0	+
Programmier-fähigkeiten	Keine	+	0	-	+	0	-	+	0	-	+	-	+	0	-			
	Wenig	0	+	0	0	+	0	0	+	0	+	0	0	+	0			
	Erfahren	0	0	+	0	0	+	0	0	+	0	+	0	0	+			

Abbildung 5-12: Passende Ausprägungskombinationen der detaillierenden Merkmale (eigene Darstellung)

Im Folgenden werden die drei Typen von Low-Code-Anwendungsfällen präsentiert, die als Grundlage für die Ableitung von Regeln für die Implementierung und den Betrieb von Low-Code-Anwendung in produzierenden Unternehmen dienen.

5.3.2 Typ I – Entwicklung durch Fachbereiche

In diesem Typ von Low-Code-Anwendungsfällen nehmen Fachbereiche eine Schlüsselposition im Entwicklungsprozess ein (siehe Abbildung 5-13). Mitarbeitende aus Fachbereichen bringen tiefgreifendes Fachwissen in spezifischen Geschäftsbereichen oder Prozessen mit und setzen Low-Code ein, um individuell zugeschnittene Anwendungen zu entwickeln. Diese Anwendungen erfüllen die speziellen Bedürfnisse ihrer Abteilungen oder Fachbereiche und sind primär für den *abteilungsinternen* Gebrauch

bestimmt. Aufgrund der begrenzten Kenntnisse in der Softwareentwicklung sind die erstellten Anwendungen für *nicht-kritische* Aufgaben vorgesehen. Sie verwalten keine kritischen oder geschäftsentscheidenden Daten oder Prozesse, wodurch ein potenzieller Ausfall der Anwendung nur einen indirekten Einfluss auf die Geschäftsprozesse hätte.

Fachbereiche dazu zu ermächtigen, ihre Anwendungen selbst zu entwickeln, kann in produzierenden Unternehmen eine Kultur der Innovation und Agilität vorantreiben. Dies ermöglicht es, schneller auf sich wandelnde Geschäftsanforderungen zu reagieren. Typischerweise sind diese Anwendungen von kurzer Lebensdauer, oft nicht länger als *wenige Jahre*, was dem dynamischen Charakter der Geschäftsanforderungen entspricht. Da die Fachexpert*innen in der Regel über *keine* bis minimale Programmierfähigkeiten verfügen, wird angestrebt, die Entwicklungsprozesse so einfach wie möglich zu gestalten. Anpassungen durch die Nutzer innerhalb der Low-Code-Plattform sind daher eher nicht vorgesehen, und die Integration mit anderen Systemen und Diensten beschränkt sich auf *standardisierte*, vorgegebene Schnittstellen.

Aufgabe	Geschäftskritikalität	Unkritisch	Kritisch	Geschäftskritisch
	Nutzung	Abteilungsintern	Abteilungsübergreifend	Standortübergreifend
	Lebenszyklus	Wenige Jahre (1-2)	Mehrere Jahre (3-5)	Viele Jahre (5-10)
Technologie	Anpassungsmöglichkeit	Low-Code		Source-Code
	Schnittstellen	Standardisiert	Konfigurierbar	Individuell
Akteur	Programmierfähigkeiten	Keine	Wenig	Erfahren

Abbildung 5-13: Typ I – Entwicklung durch Fachbereiche (eigene Darstellung)

Im Folgenden wird ein Anwendungsfallbeispiel für den zuvor beschriebenen Typ I dargestellt.

Anwendungsfallbeispiel: Dashboards für das Shopfloor Management

Im Rahmen eines Expert*inneninterviews wurde ein Typ-I-Anwendungsfall identifiziert. Dabei handelt es sich um Low-Code-Anwendung, die von Mitarbeitenden im Bereich „Operational Excellence“ bei einem deutschen Elektronikhersteller unter Verwendung von dem Microsoft Low-Code-Tool „Power BI“ entwickelt wurden. Power BI ist insbesondere für Low-Code-Entwickelnde ohne umfangreiche Programmierfähigkeiten geeignet, da die Anpassungsmöglichkeiten auf *Low-Code* begrenzt sind und sich die Nutzung auf *standardisierte Schnittstellen* konzentriert. Die entwickelten Anwendungen zielen darauf ab, Daten zu digitalisieren, um die daraus gewonnenen Informationen schneller zugängig zu machen. Durch den einfacheren Zugang zu Informationen unterstützen sie die Prozessoptimierung, ohne den Prozess selbst zu verändern und gelten somit als *unkritische* Anwendungen. Ein spezifisches Beispiel hierfür sind digitale Dashboards, die dazu dienen, analoge Anzeigetafeln auf dem Shopfloor zu ersetzen. Da diese Dashboards individuell für die jeweiligen Bereiche des Shopfloors entwickelt werden, werden die Anwendungen lediglich *abteilungsintern* genutzt.

Die Entwicklung direkt in den Fachbereichen fördert einen kontinuierlichen Feedbackprozess, der es ermöglicht, die Anwendungen laufend an veränderte Geschäftsanforderungen anzupassen. Daher wird von einer kurzen Lebensdauer der Anwendungen von *wenigen Jahren* ausgegangen. Die Verantwortung für die Entwicklung der Anwendungen liegt bei den Fachabteilungen selbst. Diese setzen dafür Mitarbeiter*innen ein, die *keine* bis *wenig Programmiererfahrung* haben, aber mit Power BI umgehen können. Aktuell sind die entwickelten Lösungen überwiegend isoliert. In Zukunft plant der Elektronikhersteller jedoch, komplexere Anwendungen zu entwickeln, die eine Verknüpfung verschiedener Fachbereiche ermöglichen sollen.

Somit illustriert dieser Anwendungsfall, wie Low-Code-Plattformen wie Power BI es Mitarbeiter*innen aus Fachbereichen ohne Programmiererfahrung ermöglichen, effektive, auf Prozessoptimierung ausgerichtete Anwendungen zu entwickeln, die nicht nur interne Prozesse optimieren, sondern auch den Grundstein für zukünftige, bereichsübergreifende Integrationen legen.

5.3.3 Typ II – Entwicklung durch Fachbereich und IT-Spezialist*innen

Dieser Anwendungsfalltyp illustriert einen kooperativen Ansatz in der Low-Code-Anwendungsentwicklung, bei dem der Fachbereich mit IT-Spezialist*innen zusammenarbeitet (siehe Abbildung 5-14). Die Fachbereiche bringen ihr tiefgehendes Verständnis für Geschäftsanforderungen ein, um den Entwicklungsprozess zu initiieren, während die IT-Spezialist*innen ihre technische Expertise einsetzen, um Unterstützung zu leisten und eine reibungslose Integration in die bestehende IT-Architektur zu gewährleisten. Diese Integration ist entscheidend, da die entwickelten Anwendungen darauf abzielen, mehrere bestehende Systeme zu vereinen, um die Zusammenarbeit *abteilungsübergreifend* zu verbessern. Aufgrund ihrer umfassenden Abdeckung und Bedeutung für die Geschäftsprozesse werden diese Anwendungen als *kritisch* eingestuft, da ein Ausfall weitreichende Folgen haben könnte. Es wird erwartet, dass diese Anwendungen eine Lebensdauer von *mehreren Jahren* haben.

Um die Fachbereiche in der Low-Code-Entwicklung zu unterstützen, sind die Anpassungsmöglichkeiten der Plattform zwar auf den *Hersteller* beschränkt, der Spielraum für Schnittstellen wird jedoch erweitert. In der Zusammenarbeit mit IT-Spezialist*innen können *konfigurierbare Schnittstellen* genutzt werden, um eine nahtlose Integration verschiedener Anwendungen zu ermöglichen, die einen abteilungsübergreifenden Mehrwert schaffen. Durch die Vereinigung der Programmierfähigkeiten von IT-Spezialist*innen mit dem fachlichen Wissen der Fachbereiche entsteht ein Mittelweg, der es ermöglicht, komplexere Anwendungen zu entwickeln, als es den Fachbereichen alleine möglich wäre. Diese synergetische Zusammenarbeit führt zur Entwicklung von umfassenden Anwendungen, die nicht nur den geschäftlichen Anforderungen gerecht werden, sondern auch den höheren Standards der IT-Governance gerecht werden.

Aufgabe	Geschäftskritikalität	Unkritisch	Kritisch	Geschäftskritisch
	Nutzung	Abteilungsintern	Abteilungsübergreifend	Standortübergreifend
	Lebenszyklus	Wenige Jahre (1-2)	Mehrere Jahre (3-5)	Viele Jahre (5-10)
Technologie	Anpassungsmöglichkeit	Low-Code		Source Code
	Schnittstellen	Standardisiert	Konfigurierbar	Individuell
Akteur	Programmierfähigkeiten	Keine	Wenig	Erfahren

Abbildung 5-14: Typ II – Entwicklung durch Fachbereiche und IT-Spezialist*innen (eigene Darstellung)

Im nachfolgenden Absatz wird ein Anwendungsfall für den zuvor erläuterten Typ II beschrieben.

Anwendungsfallbeispiel: Erstellung und Verwaltung von „Tech-Karten" für Auffälligkeiten auf dem Shopfloor

In einem weiteren Expert*inneninterview wurde ein Typ-II-Anwendungsfall aufgedeckt, der die Zusammenarbeit zwischen IT-Abteilung und Fachbereichen bei der Entwicklung von Low-Code-Anwendung bei einem Lebensmittelherstellers hervorhebt. Verwendet wird dabei die Low-Code-Plattform „Power Automate" von Microsoft, die nur *Low-Code*-Anpassungen erlaubt, aber auch die Implementierung von sowohl standardisierten als auch *konfigurierbaren Schnittstellen* unterstützt. Die IT-Spezialist*innen übernehmen eine unterstützende Funktion, besonders weil in der Vergangenheit von den Fachbereichen eigenständig entwickelte Anwendungen häufig einen unzureichend durchdachten Funktionsumfang hatten und nicht den Standards der IT-Governance entsprachen. Ein spezifisches Beispiel für den Einsatz dieses Anwendungsfalls sind die Erfassung und Verwaltung von „Tech-Karten". „Tech-Karten" werden erstellt, um Auffälligkeiten auf dem Shopfloor, wie beispielsweise unsaubere Maschinen oder fehlendes Material, zu dokumentieren und zu beseitigen. Die Anwendung wird *abteilungsübergreifend* genutzt und ist essenziell für die Aufrechterhaltung der Produktionsabläufe. Ein Ausfall der Anwendung könnte *kritische* Konsequenzen haben, welche bis zu Produktionsstopps reichen können. Die Anwendung ist bereits seit einiger Zeit im Einsatz und wird voraussichtlich *mehrere Jahre* verwendet. Das Projekt verdeutlicht, wie Power Automate es Personen mit einem grundlegenden Verständnis für Logik und *wenig Programmierfähigkeiten* ermöglicht, bedeutungsvolle Lösungen zu entwickeln.

Zusammenfassend zeigt dieser Fall beispielhaft, wie durch die kollaborative Entwicklung von Low-Code-Anwendung zwischen IT und Fachbereichen mittels Power Automate, abteilungsübergreifende Lösungen entstehen, die einen Beitrag zur Effizienzsteigerung und langfristigen Sicherheit der Produktionsprozesse leisten.

5.3.4 Typ III – Entwicklung durch IT-Spezialist*innen

In diesem speziellen Low-Code-Anwendungsfall nehmen IT-Spezialist*innen eine führende Rolle im Entwicklungsprozess der Anwendung ein (siehe Abbildung 5-15). Die

IT-Spezialist*innen wenden Low-Code an, um die Entwicklungszeiten zu verkürzen und die IT-Operationen zu verbessern. Mitarbeitende aus verschiedenen Fachbereichen und Standorten tragen zur Anforderungsdefinition bei, während die Hauptverantwortung für die Entwicklung bei den IT-Spezialist*innen verbleibt. Die Anwendungen richten sich besonders an *geschäftskritische* Abläufe und ermöglichen eine *standortübergreifende* Nutzung. Die vielfältigen Anforderungen führen zu einem hohen Entwicklungsaufwand, wodurch der Einsatz dieser Anwendungen für *viele Jahre* geplant ist. Die Vielfältigkeit der Anforderungen erfordert außerdem ein hohes Maß an Anpassungsfähigkeit und Integrationsmöglichkeiten der Low-Code-Plattform. Um alle funktionalen sowie nicht-funktionalen Anforderungen erfüllen zu können, sind Anpassungen im *Source Code* und *individuelle* Schnittstellen notwendig.

Aufgabe	Geschäftskritikalität	Unkritisch	Kritisch	Geschäftskritisch
	Nutzung	Abteilungsintern	Abteilungsübergreifend	Standortübergreifend
	Lebenszyklus	Wenige Jahre (1-2)	Mehrere Jahre (3-5)	Viele Jahre (5-10)
Technologie	Anpassungsmöglichkeit	Low-Code		Source-Code
	Schnittstellen	Standardisiert	Konfigurierbar	Individuell
Akteur	Programmierfähigkeiten	Keine	Wenig	Erfahren

Abbildung 5-15: Typ III – Entwicklung durch IT-Spezialist*innen (eigene Darstellung)

Anwendungsbeispiel: Funktionalitätserweiterung des MES

Im Rahmen eines Expert*inneninterviews mit einem deutschen Automobilhersteller wurde der dritte Anwendungstyp identifiziert. Die IT-Abteilung des Automobilherstellers verfolgt die Strategie, Standardsoftware mit Low-Code-Anwendung zu erweitern, um eine höhere Anpassungsfähigkeit zu erreichen und nicht vollständig von den Softwareherstellern abhängig zu sein. Insbesondere im Bereich des „Manufacturing Execution Systems“ (MES), das direkt auf dem Shopfloor zum Einsatz kommt, zeigt sich der Nutzen dieser Strategie. Low-Code ermöglicht es, Anwendungen für die Mitarbeitende zu entwickeln, die sie effektiver durch ihre Arbeitsprozesse leiten und wichtige Informationen direkt aus der Produktion erfassen, wie beispielsweise Rückmeldungen im Montageprozess oder Ergebnisse der Qualitätsprüfung. Ein Beispiel hierfür ist die Darstellung einer Liste von Schraubvorgängen, die mit digital vernetzten Werkzeugen durchgeführt werden sollen, wobei das MES auf Basis der Eingaben durch die Schraubtechnologie die korrekten Folgeaktionen bestimmt. Diese Flexibilität in der Anpassung der Workflow-Logik macht Low-Code besonders wertvoll. Da das MES *geschäftskritische* Produktionsprozesse abbildet und *standortübergreifend* genutzt wird, stellt die *langfristige Nutzung* dieser Anwendungen eine Priorität dar. Allerdings stößt der Automobilhersteller auf Herausforderungen bezüglich der begrenzten *Low-Code*-Anpassbarkeit, was teilweise eine Rückkehr zu traditionellem Coding erforderlich machte. Daher wird empfohlen, eine Low-Code-Plattform zu wählen, die es Entwicklern ermöglicht, Code-Bausteine im *Source Code* anzupassen und *individuelle Schnittstellen* zu erstellen. Für die erfolgreiche Umsetzung dieser Anwendungen ist

tiefgehende IT-Expertise unerlässlich, insbesondere im Bereich der Schnittstellentechnologien, Datenbanken und Datenmodellierung.

Zusammenfassend zeigt dieser Anwendungsfall auf, dass die Erweiterung von Standardsoftware durch Low-Code-Anwendung im Automobilsektor eine flexible und effektive Strategie darstellt, um geschäftskritische Prozesse anzupassen und zu optimieren. Dabei ist es essenziell, eine Plattform zu wählen, die eine hohe Anpassungsfähigkeit bietet und gleichzeitig das tiefe technische Verständnis der IT-Expert*innen erfordert, um die Potenziale von Low-Code voll auszuschöpfen und langfristigen Erfolg zu sichern.

5.4 Reflexion und Zusammenfassung der Ergebnisse

Auf der Grundlage, der in Kapitel 5.3 vorgenommenen Typologie zum Einsatz von Low-Code, wurden drei kohärente Typen von Low-Code-Anwendungsfällen identifiziert: Typ I sieht die Low-Code-Entwicklung durch die Fachbereiche vor, bei Typ II erfolgt die Entwicklung durch Fachbereich und IT-Spezialist*innen und bei Typ III entwickeln die IT-Spezialist*innen die Low-Code-Anwendung. Die Typologie basiert zunächst auf dem Zweck der Typisierung (siehe Kapitel 5.1.1), gefolgt von der Bestimmung detaillierter Merkmale (siehe Kapitel 5.1.2) und ihrer Ausprägungen (siehe Kapitel 5.2.). Die so identifizierten Merkmale dienten als Grundlage für die Erstellung der Typisierung, die anschließend durch Anwendungsbeispiele untermauert wurden. Die drei abgeleiteten Typen sind in Abbildung 5-16 schematisch visualisiert.

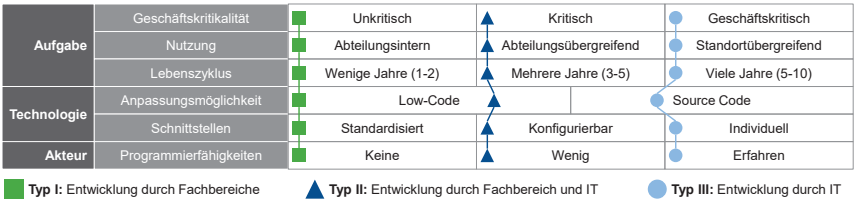

Abbildung 5-16: Typologie der Low-Code-Anwendungsfälle (eigene Darstellung)

Es ist besonders hervorzuheben, dass innerhalb einer Organisation verschiedene Formen der identifizierten Anwendungsfälle gleichzeitig existieren können. So könnten beispielsweise Anwendungen, die von Fachbereichen mit Low-Code für interne Abteilungsprozesse entwickelt wurden, parallel zu Anwendungen zum Einsatz kommen, die von Fachbereichen und IT-Spezialist*innen für standortweite Zwecke oder ausschließlich von IT-Spezialist*innen für standortübergreifende Anwendungen erstellt wurden. Darüber hinaus ist anzumerken, dass die in der Praxis vorkommenden Anwendungsfälle in Bezug auf einzelne Merkmalsausprägungen von den hier identifizierten Typen abweichen können. Die hier identifizierten Typen stellen eine Handlungsempfehlung für Unternehmen dar. Es wird empfohlen, in den nächsthöheren Anwendungsfalltyp zu wechseln, sobald eine oder mehrere der Ausprägungen des Anwendungsfalls dem höheren Typ entsprechen. Wie im Anwendungsbeispiel für Typ III deutlich wurde, entsprach eine der Anpassungsmöglichkeit nicht Typ III (Source Code), sondern Typ II

(Low-Code). Gerade diese Abweichung stellt die größten Herausforderungen im Betrieb der entwickelten Low-Code-Anwendung dar. Somit sind die identifizierten Typen eher als Handlungsempfehlungen zu verstehen, die Unternehmen beachten sollten, sobald die Entwicklung mit Low-Code in Betracht gezogen wird. Sie dient dem Zweck, das Verständnis für den möglichen Umfang von Low-Code-Anwendungen zu stärken und Transparenz über die notwendigen Fähigkeiten der Low-Code-Entwickelnden sowie die Funktionalitäten der Low-Code-Plattform zu schaffen.

Diese Ausarbeitung adressiert die in Kapitel 3.3 identifizierte Forschungslücke und beantwortet die erste spezifische und richtungsweisende Forschungsfrage:

Wie können Low-Code-Anwendungsfälle in produzierenden Unternehmen beschrieben werden?

Die vorgenommene Typisierung allein bietet jedoch keine umfassende Erklärung dafür, wie die verschiedenen Typen von Anwendungsfällen in die bestehenden Unternehmensstrukturen integriert werden können. Deshalb wird im folgenden Kapitel untersucht, welche Regeln beim Einsatz von Low-Code zu berücksichtigen sind, um anschließend spezifische Zusammenhänge zwischen den Anwendungsfalltypen und den zu beachtenden Regeln herzuleiten.

6 Beschreibung der Regeln für Low-Code

Nach der Festlegung der Low-Code-Anwendungsfalltypen konzentriert sich dieses Kapitel auf die Darstellung und Erläuterung der Regeln, die bei der Implementierung und dem Betrieb von Low-Code-Anwendung berücksichtigt werden müssen. Ziel ist es, die zweite spezifische Forschungsfrage zu beantworten:

Wie können Regeln für Low-Code innerhalb eines produzierenden Unternehmens beschrieben werden?

Der Schwerpunkt liegt darauf, Regeln für den Einsatz von Low-Code zu erarbeiten und diese in einem strukturierten Regelwerk zusammenzufassen. Das Vorgehen für das zweite Beschreibungsmodell basiert auf der strukturierten Inhaltsanalyse nach MAYRING (siehe Kapitel 4.2.5, siehe Abbildung 6-1).

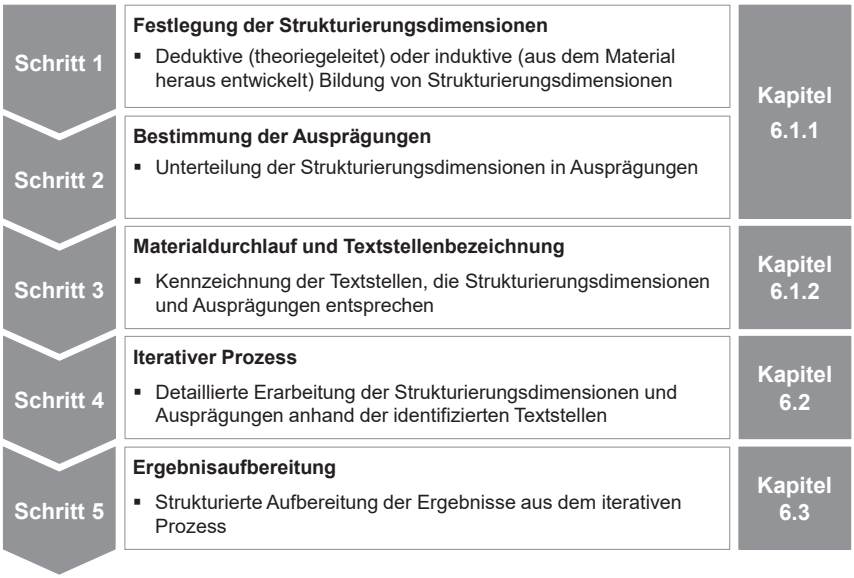

Abbildung 6-1: Vorgehensweise der strukturierten Inhaltsanalyse (eigene Darstellung i. A. a. MAYRING 1984, S. 170)

Basierend auf dem COBIT-Kernmodell werden zunächst die Strukturierungsdimensionen für das Regelwerk definiert und die Ausprägungen für jede Strukturierungsdimension in Form von Regeln bestimmt (siehe Kapitel 6.1). Anschließend werden die Regeln mithilfe einer Literaturrecherche und Textstellenbezeichnung spezifiziert (siehe Kapitel 6.2). Abschließend werden die Ergebnisse zusammengefasst und reflektiert (siehe Kapitel 6.3). Diese methodische Herangehensweise ermöglicht eine strukturierte Darstellung der Regeln für den Einsatz von Low-Code in einem systematischen Regelwerk.

6.1 Definition der Low-Code-Regeln

Die Identifikation der Low-Code-Regeln erfolgt nach Schritt eins bis drei der Inhaltanalyse von MAYRING (siehe Abbildung 6-1). Dazu werden zunächst Strukturierungsdimensionen basierend auf den Domänen des COBIT-Kernmodells definiert und die zugehörigen Low-Code-Regeln basierend auf den Zielen des COBIT-Kernmodells festgelegt (siehe Kapitel 6.1.1). Daraufhin erfolgt eine strukturierte Literaturrecherche, in deren Rahmen relevante Textstellen markiert werden, die Aufschluss über die zuvor definierten Low-Code-Regeln geben. Die markierten Textstellen werden anschließend bewertet und den entsprechenden Low-Code-Regeln zugeordnet (siehe Kapitel 6.1.2). Abbildung 6-2 zeigt den Zusammenhang zwischen den Domänen und Zielen des COBIT-Kernmodells, den Strukturierungsdimensionen und Low-Code-Regeln sowie den Textstellen aus der Literatur.

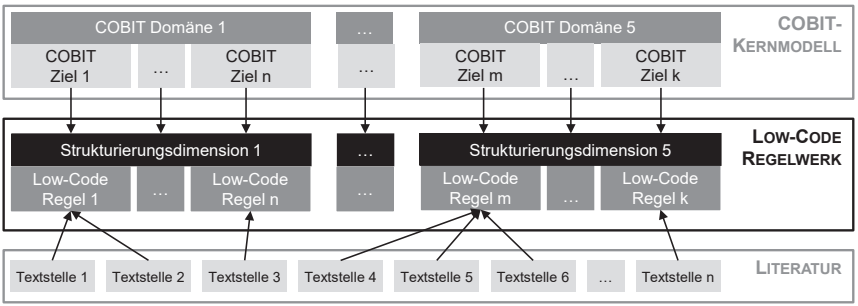

Abbildung 6-2: Zusammenhang Strukturierungsdimensionen, Regeln und Textstellen (eigene Darstellung)

6.1.1 Strukturierungsdimensionen und Regeln

Zur Definition der Strukturierungsdimensionen und der Low-Code-Regeln wird in dieser Arbeit auf das IT-Governance-Rahmenwerk COBIT 2019 zurückgegriffen (s. ISACA 2020). Das COBIT-Kernmodell, welches bereits in Kapitel 3.2.3 beschrieben wurde, deckt sowohl strategische als auch operative Aspekte der IT-Governance ab und bietet einen starken Praxisbezug. Das COBIT-Kernmodell gliedert sich in die fünf Domänen „Vorgeben“ (EDM – Evaluieren, Vorgeben, Überwachen), „Planen“ (APO – Anpassen, Planen und Organisieren), „Aufbauen“ (BAI – Aufbauen, Beschaffen und Implementieren), „Betreiben“ (DSS – Bereitstellen, Betreiben und Unterstützen) und „Überwachen“ (MEA – Überwachen, Evaluieren und Beurteilen), die jeweils vier bis vierzehn spezifische Ziele umfassen (siehe Abbildung 6-3).

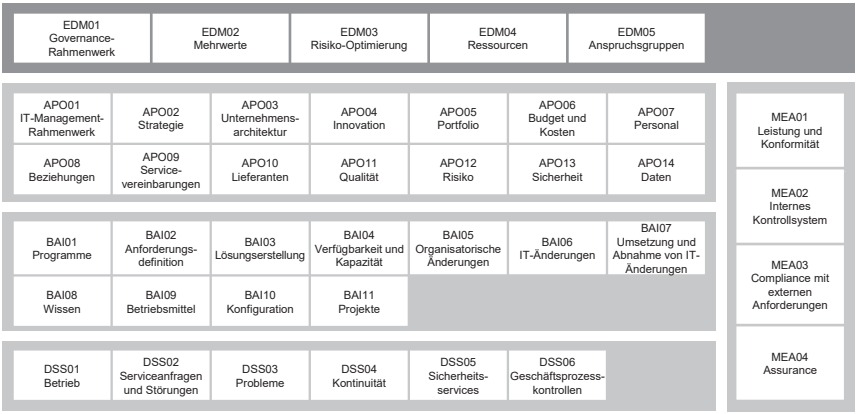

Abbildung 6-3: Ziele im COBIT-Kernmodell (eigene Darstellung i. A. a. ISACA 2020, S. 23)

In dieser Arbeit werden die fünf COBIT-Domänen als Strukturierungsdimensionen herangezogen, da die Domänen einen umfassenden, systematischen Rahmen für die Entwicklung eines Low-Code-Regelwerks darstellen (siehe Kapitel 3.2.3). Die COBIT-Ziele innerhalb jeder Domäne dienen als Low-Code-Regeln. Das Hauptziel der Analyse ist es, die relevantesten COBIT-Ziele für den Einsatz von Low-Code zu identifizieren und daraus spezifische Regeln für Low-Code-Anwendung abzuleiten. Im Folgenden werden die Strukturierungsdimensionen und Low-Code-Regeln gemäß dem COBIT-Kernmodell detailliert beschrieben und auf den Fokus der vorliegenden Arbeit eingegrenzt. Eine strukturierte Analyse der COBIT-Kernmodell-Ziele findet sich in Anhang A.1.

Vorgeben

Bei der Domäne "Vorgeben" (EDM – Evaluieren, Vorgeben und Überwachen) geht es darum, den Nutzen der IT zu maximieren, Ressourcen effizient einzusetzen, Risiken zu minimieren und die Interessen aller relevanten Anspruchsgruppen zu berücksichtigen (s. GAULKE 2019). Abbildung 6-4 zeigt, welche der COBIT-Ziele für das Low-Code-Regelwerk berücksichtigt wurden (vgl. A.1).

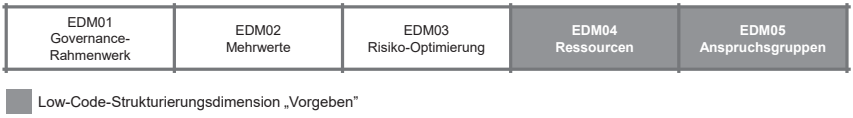

Abbildung 6-4: Relevante COBIT-Ziele für den Einsatz von Low-Code in der COBIT-Domäne „Vorgeben" (eigene Darstellung)

In der Domäne „Vorgeben" ergibt sich eine gezielte Fokussierung auf die Elemente, die für die Entwicklung und den Einsatz von Low-Code von Bedeutung sind. Es geht nicht um das gesamte Unternehmen und dessen *Governance-Rahmenwerk* sondern

speziell darum, notwendige *Ressourcen* bereitzustellen und die Interessen der *Anspruchsgruppen* zu berücksichtigen. Die COBIT-Ziele *Risiko-Optimierung* und *Mehrwerte* werden nicht als eigene Regeln berücksichtigt, da das gesamte Regelwerk darauf abzielt Mehrwerte zu schaffen und das Risiko gering zu halten (siehe Kapitel 3.2.3). Daher werden die COBIT-Ziele „Governance-Rahmenwerk“, „Mehrwerte“ und „Risiko-Optimierung“ im Folgenden nicht weiter betrachtet. Die COBIT-Ziele „Ressourcen“ und „Anspruchsgruppen“ finden sich als Low-Code-Regeln in der Strukturierungsdimension „Vorgeben“ im Regelwerk wieder.

Planen

Die Domäne „Planen" (APO – Align, Plan and Organize) spielt eine zentrale Rolle im IT-Management, indem sie sicherstellt, dass IT-Initiativen und -Ressourcen präzise auf die Unternehmensziele abgestimmt sind. Diese Abstimmung beinhaltet das Management und die Organisation von IT-Aktivitäten zur optimalen Unterstützung der Geschäftsziele. Nach GAULKE fördert eine klar strukturierte Planungsdomäne den Einsatz von IT als effektives Werkzeug zur Erreichung strategischer Unternehmensziele.

Beim Einsatz von Low-Code kann die Bedeutung einzelner COBIT-Ziele innerhalb der „Planen"-Domäne variieren. Aufgrund des spezifischen und oft begrenzteren Anwendungsbereichs von Low-Code im Vergleich zur gesamten Unternehmens-IT können einige Ziele zusammengefasst oder als weniger relevant betrachtet werden (siehe Abbildung 6-5, vgl. A.1).

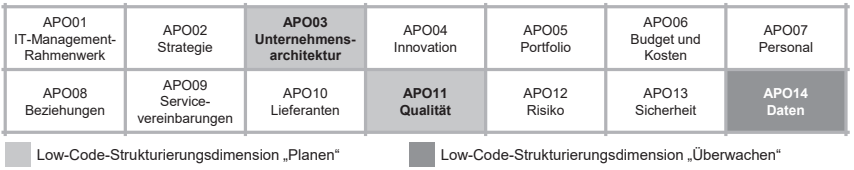

Abbildung 6-5: Relevante Ziele für den Einsatz von Low-Code in der COBIT-Domäne „Planen“ (eigene Darstellung)

Die Fokussierung auf Low-Code bedeutet, dass nicht das gesamte Unternehmen betrachtet wird, sondern primär die Unternehmensbereiche, in denen Low-Code-Anwendung eingesetzt werden. Die Entwicklung eines vollständigen *IT-Management-Rahmenwerks* oder die detaillierte Untersuchung des gesamten *Portfolios* ist daher nicht erforderlich. Zudem wird vorausgesetzt, dass die Entscheidung für die Einführung von Low-Code bereits *strategisch* getroffen wurde. Die Auswahl eines bestimmten Low-Code-Plattformanbieters und die damit verbundenen *Kostenaspekte* liegen außerhalb des Schwerpunkts dieser Arbeit (siehe Kapitel 2.3.1). Das Ziel der *Unternehmensarchitektur* wird basierend auf der Abgrenzung der Untersuchung in Kapitel 2.3.1 auf IT-Architektur eingegrenzt. In dieser Arbeit wird untersucht, wie neue Low-Code-Plattformen in die bestehende IT-Architektur integriert werden können und wie die *Qualität* der Low-Code-Anwendung sichergestellt werden kann. Zusätzlich soll durch das *Überwachen* der *Daten* die Compliance gewährleistet werden. Folglich werden die COBIT-

Ziele „Unternehmensarchitektur“ und „Qualität“ als Low-Code-Regeln für „IT-Architektur“ und „Qualität“ in der Strukturierungsdimension „Planen“ im Regelwerk verortet, während sich das COBIT-Ziel „Daten“ als Low-Code-Regel in der Dimension „Überwachen“ wiederfindet.

Aufbauen

Die Domäne „Aufbauen" (BAI – Build, Acquire and Implement) ist entscheidend für die Umsetzung der IT-Strategie eines Unternehmens. In diesem Stadium werden zunächst potenzielle Anwendungen ermittelt, die entweder intern entwickelt oder von externen Anbietern bezogen werden (siehe Kapitel 2.2.2). Anschließend erfolgt die Integration der Anwendungen in die bestehende Unternehmensarchitektur. Dieser Prozess beinhaltet die Implementierung, Konfiguration und Anpassung der Anwendungen, um eine reibungslose Interaktion innerhalb der Organisation zu ermöglichen und den beabsichtigten Mehrwert zu realisieren. (s. GAULKE 2019)

In Bezug auf Low-Code-Anwendung sind innerhalb dieser Domäne spezifische Ziele von besonderer Bedeutung (siehe Abbildung 6-6, vgl. A.1).

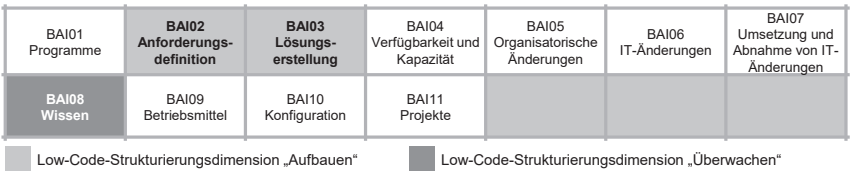

Abbildung 6-6: Relevante Ziele für den Einsatz von Low-Code in der COBIT-Domäne „Aufbauen“ (eigene Darstellung)

Bei der Eingrenzung auf Low-Code werden identifizierte potenzielle Anwendungen basierend auf ihrer *Anforderungsdefinition* darauf hin geprüft, ob sie mittels Low-Code umsetzbar sind. Dieser Fokus bedeutet, dass eine umfassende Berücksichtigung des gesamten *Programmmanagements* oder der *Betriebsmittel* nicht erforderlich ist. Stattdessen verlagert sich der Schwerpunkt auf die *Lösungserstellung*. In diesem Stadium werden die Low-Code-Anwendungen direkt nutzbar gemacht. Es sollte außerdem kontinuierlich *überwacht* werden, dass das *Wissen* über die Low-Code-Anwendungen zur Verfügung steht. Demnach finden sich die COBIT-Ziele „Anforderungsdefinition“ und „Lösungserstellung“ als Low-Code-Regeln in der Strukturierungsdimension „Aufbauen“ wieder, während das COBIT-Ziel „Wissen“ in der Dimension „Überwachen“ verortet ist.

Ausführen

Die Domäne „Ausführen" (DSS – Deliver, Service and Support) ist für die Gewährleistung eines stabilen und zuverlässigen IT-Betriebs innerhalb eines Unternehmens verantwortlich. Sie fokussiert sich auf die kontinuierliche Verbesserung der IT-Services und bietet Unterstützung für die Anwender. Die effiziente Reaktion auf Störungen und Probleme sichert die Verlässlichkeit der IT-Dienste und garantiert, dass diese einen

signifikanten Beitrag zur Gesamtleistung des Unternehmens erbringen. (s. GAULKE 2019, s.)

Abbildung 6-7 zeigt die spezifischen Aspekte dieser Domäne, die in Bezug auf den Einsatz von Low-Code von Bedeutung sind (vgl. A.1).

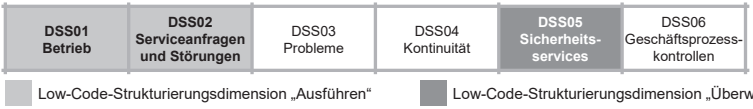

Abbildung 6-7: Relevante Ziele für den Einsatz von Low-Code in der COBIT-Domäne „Ausführen“ (eigene Darstellung)

Die Konzentration auf Low-Code innerhalb dieser Domäne bedeutet, dass nicht das gesamte Spektrum der IT-Services betrachtet wird, sondern lediglich der *Betrieb* und der Support in Form von *Serviceanfragen und Störungen* beim Ausführen von Low-Code-Anwendung von Bedeutung sind, da diese Services direkt in Verbindung mit den Low-Code-Anwendung stehen. Darüber hinaus sollte die IT-Sicherheit der Low-Code-Plattform und Anwendungen durch das *Überwachen*, unter anderem in Form von *Sicherheitsservices* (siehe Kapitel 2.3.5), gewährleistet werden. Folglich werden die COBIT-Ziele „Betrieb“ und „Serviceanfragen und Störungen“ zu den Low-Code-Regeln für „Betrieb“ und „Support“ in der Strukturierungsdimension „Ausführen“ verortet, während sich das COBIT-Ziel „Sicherheitsservices“ als Low-Code-Regel zur „IT-Sicherheit“ in der Dimension „Überwachen“ wiederfindet.

Überwachen

Die Domäne „Überwachen" (MEA – Monitor, Evaluate and Assess) ist darauf ausgerichtet, die IT-Lösungen eines Unternehmens kontinuierlich zu überwachen, um sicherzustellen, dass sie den geschäftlichen Anforderungen entsprechen. Das primäre Ziel dieser Domäne ist es, die Leistung und Effektivität der IT-Services ständig zu beurteilen und zu optimieren. Durch regelmäßige Überprüfung und Bewertung lassen sich Schwachstellen aufdecken und korrigieren, was wiederum zur Sicherung einer hohen Servicequalität und Kundenzufriedenheit beiträgt. Ferner ermöglicht die Beurteilung von Verbesserungsmaßnahmen, dass die IT-Organisation flexibel und schnell auf Veränderungen in den Geschäftsbedingungen und technologischen Entwicklungen reagieren kann. (s. GAULKE 2019)

Bei der Anwendung dieser Domäne auf Low-Code-Plattformen erscheinen die Ziele „Leistung und Konformität", „Internes Kontrollsystem", „Compliance mit externen Anforderungen" und „Assurance" allerdings als zu allgemein (siehe Abbildung 6-8). Sie liefern gute Ansätze, treffen aber auf die spezifischen Überwachungsanforderungen von Low-Code nur bedingt zu.

MEA01 Leistung und Konformität	MEA02 Internes Kontrollsystem	MEA03 Compliance mit externen Anforderungen	MEA04 Assurance

Abbildung 6-8: COBIT-Ziele für den Einsatz von Low-Code in der Domäne „Überwachen“ (eigene Darstellung)

Daher ist es sinnvoll, spezifischere Ziele aus anderen Domänen zu übernehmen und diese an die Low-Code-Anforderungen anzupassen, um eine effektive und zielgerichtete Überwachung der mit Low-Code entwickelten Anwendungen zu ermöglichen. Aus diesem Grund werden die COBIT-Ziele „Daten“ aus der Domäne „Planen“, „Wissen“ aus der Domäne „Aufbauen“ sowie das COBIT-Ziel „Sicherheitsservices“ aus der Domäne „Ausführen“ zu den Low-Code-Regeln „Daten“, „Wissen“ und „IT-Sicherheit“ in die Strukturierungsdimension „Überwachen“ ins Regelwerk überführt.

Zusammenfassung

Abbildung 6-9 zeigt die identifizierten Strukturierungsdimensionen basierend auf den fünf COBIT-Domänen und die Low-Code-Regeln, die aus ausgewählten COBIT-Zielen abgeleitet wurden.

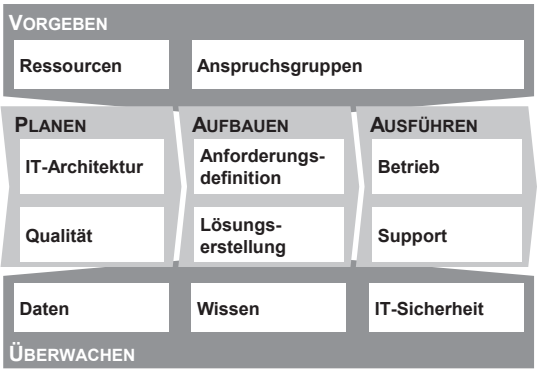

Abbildung 6-9: Low-Code-Strukturierungsdimensionen und Regeln (eigene Darstellung)

Innerhalb der Dimension „Vorgeben“ konzentrieren sich die Regeln auf das Bereitstellen von Ressourcen und das Zusammenbringen der Anspruchsgruppen. In der Dimension „Planen“ sind die Regeln auf die Integration der Low-Code-Plattform in die IT-Architektur und auf die Gewährleistung der Qualität der Anwendungen fokussiert. Die Dimension „Aufbauen“ umfasst die Anforderungsdefinition und die Lösungserstellung der Low-Code-Anwendung, während sich die Dimension „Ausführen“ auf den Betrieb und den Support der Anwendungen konzentriert. In der Dimension „Überwachen“ werden die Regeln zur Überwachung und Verwaltung der Daten sowie zur Sicherstellung des Wissensmanagements und der IT-Sicherheit zusammengefasst.

6.1.2 Low-Code-Textstellen

Im Folgeschritt wird die Definition der bereits identifizierten Regeln verfeinert und auf den Low-Code-Kontext spezifischer angewendet. In Kapitel 3.2.1 erfolgte bereits eine strukturierte Literaturrecherche, um Regeln für den Einsatz von Low-Code zu identifizieren. Die Anzahl der auf diese Weise gefundenen Quellen erwies sich jedoch als unzureichend. Deshalb wurde eine zusätzliche Literaturrecherche durchgeführt, die darauf abzielt, notwendige Regeln für den Einsatz von Low-Code zu identifizieren, indem sie die mit Low-Code assoziierten Herausforderungen und Risiken untersuchte. Auf Grundlage der erkannten Risiken und Herausforderungen lassen sich folglich gezielt Regeln ableiten, die darauf ausgerichtet sind, den Risiken und Herausforderungen entgegenzuwirken. Tabelle 17 fasst die untersuchten Quellen für die Erarbeitung der Low-Code-Regeln zusammen.

Tabelle 17: Untersuchte Quellen für die Identifikation von Low-Code-Regeln (eigene Darstellung)

Lfd. Nr.	Untersuchte Quelle	Titel
1	Al Alamin et al. 2020	An Empirical Study of Developer Discussions on Low-Code-Software Development Challenges
2	Benac u. Mohd 2022	Recent Trends in Software Development: Low-Code Solutions
3	Binzer u. Winkler 2022	Democratizing Software Development: A Systematic Multivocal Literature Review and Research Agenda on Citizen Development
4	Di Ruscio et al. 2022	Low-code development and model-driven engineering: Two sides of the same coin?
5	Elshan et al. 2023	An Investigation of Why Low Code Platforms Provide Answers and New Challenges
6	Escalona et al. 2022	A Quantitative SWOT-TOWS Analysis for the Adoption of Model-Based Software Engineering
7	Heuer et al. 2022	Towards a Governance of Low-Code Development Platforms Using the Example of Microsoft PowerPlatform in a Multinational Company
8	Ihirwe et al. 2021	Cloud-based modeling in IoT domain: a survey, open challenges and opportunities
9	Ihirwe et al. 2020	Low-code Engineering for Internet of things: A state of research
10	Indamutsa et al. 2021	A Low-Code Development Environment to Orchestrate Model Management Services
11	Khorram et al. 2020	Challenges & Opportunities in Low-Code Testing
12	Lethbridge 2021	Low-Code Is Often High-Code, So We Must Design Low-Code Platforms to Enable Proper Software Engineering
13	Luo et al. 2021	Characteristics and Challenges of Low-Code Development: The Practitioners Perspective
14	Overeem et al. 2021	API Management Maturity of Low-Code Development Platforms
15	Prinz et al. 2022	Two Perspectives of Low-Code Development Platform Challenges – An Exploratory Study
16	Rogozov et al. 2023	Approach to the Implementation of Intelligent Low-Code Platforms
17	Rokis u. Kirikova 2022	Challenges of Low-Code/No-Code Software Development: A Literature Review

Insgesamt wurden 17 relevante Quellen hinsichtlich der Erwähnung von Risiken, Herausforderungen und Regeln im Kontext von Low-Code analysiert. Die in diesen Quellen identifizierten Textstellen wurden den vorab definierten Low-Code-Regeln zugeordnet. Diese systematische Zuordnung ermöglicht eine detaillierte und vergleichbare Analyse der Inhalte jeder Quelle. Dadurch wird nicht nur der Fokus jeder untersuchten Quelle klar herausgearbeitet, sondern auch eine präzisere und praxistaugliche Definition von Regeln für ein Low-Code-Regelwerk erreicht. Die Zuweisung der Quellen zu

den jeweiligen Regeln wird in Abbildung 6-10 visualisiert. Auf Basis dieser Erkenntnisse werden im nachfolgenden Kapitel spezifische Definitionen für die einzelnen Low-Code-Regeln dargelegt.

		Quellen																
Strukturierungs-dimensionen	Regeln	Alamin 2021	Benac 2022	Binzer 2022	Di Ruscio 2022	Elshan 2023	Escalona 2022	Heuer 2022	Indamutsa 2021	Ihirwe 2020	Ihirwe 2020	Khorram 2020	Lethbridge 2021	Luo 2021	Overeem 2021	Prinz 2022	Rokis 2022	Rogozov 2023	Summe
Vorgeben	Ressourcen			x				x	x	x				x		x		x	7
	Anspruchsgruppen				x	x		x								x			4
Planen	IT-Architektur	x		x				x							x	x	x		6
	Qualität			x		x		x	x				x			x			6
Aufbauen	Anforderungsdefinition					x	x	x						x		x	x		6
	Lösungserstellung	x		x		x	x		x	x	x	x		x		x	x	x	12
Ausführen	Betrieb				x		x	x	x	x			x	x		x	x		8
	Support						x									x			2
Überwachen	Daten					x		x								x			3
	Wissen		x			x		x	x				x			x			6
	IT-Sicherheit			x				x	x	x						x			5

Abbildung 6-10: Zuweisung der Quellen zu den Low-Code-Regeln (eigene Darstellung)

6.2 Spezifikation der Low-Code-Regeln

Im Folgenden werden die Low-Code-Regeln anhand der fünf Dimensionen „Vorgeben“, „Planen“, „Aufbauen“, „Ausführen“ und „Überwachen“ strukturiert und vorgestellt. Abbildung 6-11 zeigt die detaillierte Übersicht der definierten Low-Code-Regeln.

Abbildung 6-11: Strukturierte Übersicht der Regeln für den Einsatz von Low-Code (eigene Darstellung)

6.2.1 Vorgeben

In der Dimension „Vorgeben“ werden die Rahmenbedingungen definiert, welche die Ausgestaltung des unternehmensweiten Einsatzes von Low-Code-Plattformen vorgeben. Diese Vorgaben gewährleisten, dass der Gebrauch von Low-Code im Einklang mit den übergeordneten Unternehmenszielen und -standards erfolgt. Innerhalb dieser Dimension konzentrieren sich die Regeln auf das Bereitstellen von Ressourcen und das Zusammenbringen der Anspruchsgruppen.

Ressourcen

Im Vordergrund der Regel „Ressourcen“ steht die effiziente Nutzung der Ressourcen, die sich mit der Entwicklung von Low-Code-Anwendung auseinandersetzen sollen.

Die Forschungsliteratur betont die Wichtigkeit, ausreichende Kapazitäten bereitzustellen und die involvierten Mitarbeitende angemessen zu schulen, um Low-Code-Plattformen erfolgreich einzusetzen. Obwohl Low-Code den Einstieg in die Softwareentwicklung erleichtert, sind grundlegende Programmierfähigkeiten nach wie vor unerlässlich (s. Luo et al. 2021; Rogozov et al. 2023; Heuer et al. 2022; Ihirwe et al. 2021; Binzer u. Winkler 2022). Ein solides Verständnis für Programmierung ist notwendig, da die intuitive Nutzung von Low-Code-Plattformen nicht immer gegeben ist (s. Rokis u. Kirikova 2022). Low-Code-Entwickelnde müssen daher nicht nur die Perspektive der Endbenutzer*innen verstehen, sondern auch die von Softwareentwickelnden einnehmen können, um Lösungen zu entwickeln, die Geschäftsprobleme effektiv adressieren (s. Rogozov et al. 2023). Daher ist es entscheidend, dass Low-Code-Entwickelnde darin geschult werden, eine Entwicklerperspektive einzunehmen. Dies erfordert ein spezifisches Schulungskonzept in Kombination mit einem Rollenmodell. Innerhalb des Rollenmodells können die erforderliche Kapazität und der Schulungsbedarf für jede Rolle festgelegt werden. Es ist die Aufgabe des Managements, für ausreichende Kapazitäten und angemessene Schulungen zu sorgen (s. Prinz et al. 2022; Elshan et al. 2023). Aus diesem Grund wird folgende Regel definiert:

Regel: Rollenmodell mit notwendigen Kapazitäten und Schulungen definieren

In der praktischen Umsetzung dieser Regel liegt der Fokus auf der koordinierten Kapazitätsbereitstellung aus den IT-Abteilungen und Fachbereichen (s. Heuer et al. 2022). Die Kombination von technischem Fachwissen und Domänenkenntnissen ist für eine optimale Ressourcennutzung entscheidend. Die involvierten Fachbereiche sollten aktiv in die Entwicklung einbezogen werden und nicht nur als Nutzer fungieren (s. Prinz et al. 2022; Elshan et al. 2023). Durch die Einführung eines Rollenmodells (s. Prinz et al. 2022; Indamutsa et al. 2021) wird die Kapazitätsauslastung der Low-Code-Entwickelnden festgelegt und die erforderlichen Fähigkeiten definiert. Gezielte Schulungen stärken das Know-how der Mitarbeitenden und erweitern ihre Fähigkeiten in der Problemlösung und Anwendungsentwicklung mit Low-Code.

Zusammengefasst ermöglicht das Management durch ein Rollenmodell inklusive notwendiger Kapazitäten und Schulungsbedarf eine effiziente Nutzung von Low-Code-

Ressourcen. Die Kombination von Kapazitätsbereitstellung, Rollenmodell und gezielten Schulungen fördert eine effiziente Entwicklung mit Low-Code.

Anspruchsgruppen

Der Schwerpunkt der Regel „Anspruchsgruppen" liegt auf dem Aufbau einer Organisationsstruktur, welche die Zusammenarbeit und die Beziehungen zwischen den verschiedenen Beteiligten aus der IT und den Fachbereichen fördert.

ELSHAN ET AL. empfehlen in diesem Kontext den Aufbau eines „Center of Excellence". Diese zentrale Einrichtung vereint die zuvor definierten Rollen, um eine effizientere Zusammenarbeit zu ermöglichen und fördert den gemeinschaftlichen Ansatz zur Lösung von Problemen in der Low-Code-Softwareentwicklung. Innerhalb dieses Centers sollen nicht nur eine Plattform für den Austausch von Informationen und gegenseitige Unterstützung geschaffen werden, sondern auch gemeinschaftlich Entscheidungen getroffen werden. Eine entscheidende Voraussetzung für die erfolgreiche Implementierung organisatorischer Veränderungen ist die aktive Unterstützung durch das Top-Management (s. BINZER U. WINKLER 2022). Daher wird folgende Regel festgelegt:

Regel: Center of Excellence zum Erfahrungsaustausch und Entscheidungen treffen einführen

Innerhalb des Centers sollen bewährte Verfahren und Erfahrungen diskutiert werden (s. HEUER ET AL. 2022) und notwendige Entscheidungen getroffen werden. Eine optimale Zusammenarbeit wird erreicht, indem die Fachbereiche bei der Low-Code-Entwicklung nicht nur frühzeitig und kontinuierlich in die Erfassung von Anforderungen einbezogen werden, sondern auch aktiv in die Entwicklung einbezogen werden und bei Entscheidungsprozessen berücksichtigt werden (s. HEUER ET AL. 2022). Das „Center of Excellence" strebt eine Kultur an, in der die Beteiligten proaktiv Probleme lösen und Verbesserungen anstreben (s. BINZER U. WINKLER 2022). Dies fördert einen positiven Ansatz zur Zusammenarbeit und sichert den Erfolg einer langfristigen Kooperation (s. PRINZ ET AL. 2022). Außerdem kann das Center of Excellence dazu beitragen, Beteiligte vom Nutzen von Low-Code zu überzeugen und den Widerständen gegen die damit einhergehenden Änderungen entgegenzuwirken (s. ELSHAN ET AL. 2023; PRINZ ET AL. 2022). So kann auf lange Sicht ein kontinuierliches Wachstum der Nutzerbasis gewährleistet werden.

Zusammenfassend fördert die Einrichtung eines „Center of Excellence" die Zusammenarbeit zwischen den Geschäftsbereichen und der IT, indem es die definierten Rollen kontinuierlich zusammenbringt. Dies schafft eine Umgebung, die eine Kultur von proaktiven Beteiligten begünstigt und unterstützt.

Abbildung 6-12 fasst die Low-Code-Regeln in der Dimension „Vorgeben" zusammen. Innerhalb dieser Dimension konzentrieren sich die Regeln auf das Bereitstellen von Ressourcen und das Zusammenbringen der Anspruchsgruppen.

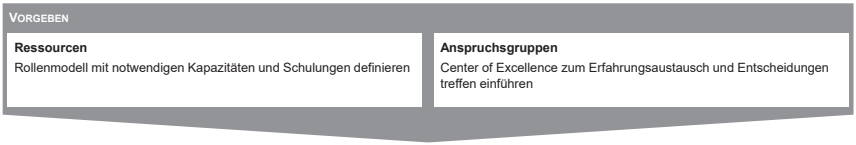

Abbildung 6-12: Low-Code-Regeln in der Dimension „Vorgeben" (eigene Darstellung)

6.2.2 Planen

In der Dimension „Planen“ wird die Integration der Low-Code-Entwicklungsumgebungen in die bestehende IT-Architektur festgelegt, um die mit Low-Code erstellten Anwendungen optimal einzubinden. Zudem wird ein Vorgehen definiert, um bei den entwickelten Anwendungen die geforderte Qualität sicherstellen zu können. Innerhalb dieser Dimension konzentrieren sich die Regeln auf die Integration der Low-Code-Plattform in die IT-Architektur und die Gewährleistung der Qualität der Anwendungen.

IT-Architektur

Die Regel „IT-Architektur" betrifft insbesondere die Integration von Low-Code-Plattformen in die vorhandene IT-Architektur (siehe Kapitel 2.2.3).

Die Interoperabilität zwischen bestehenden IT-Systemen und Low-Code-Plattformen nimmt eine zentrale Rolle ein, um die Leistungsfähigkeit und Effektivität der entwickelten Anwendungen zu sichern (s. PRINZ ET AL. 2022; ROKIS U. KIRIKOVA 2022; IHIRWE ET AL. 2021). Daher ist die reibungslose Integration der Low-Code-Plattform in die vorhandene Technologiearchitektur von großer Bedeutung. Innerhalb des Integrationsprozesses treten verschiedene Herausforderungen auf, darunter die begrenzte Verfügbarkeit passender und einsatzbereiter Integrationswerkzeuge (s. ESCALONA ET AL. 2022) und das Fehlen von sofort verfügbaren standardisierten Schnittstellen (s. BINZER U. WINKLER 2022; PRINZ ET AL. 2022; ROKIS U. KIRIKOVA 2022; LUO ET AL. 2021; ESCALONA ET AL. 2022). Eine Lösung kann die Konfiguration eigener Schnittstellen bieten, obwohl dies oft mit Problemen verbunden ist, die den Integrationsprozess weiter erschweren (s. ROKIS U. KIRIKOVA 2022). Ferner stellt die Nutzung externer Webanfragen, um Daten oder Services von einem externen Server über das Internet zu erhalten, eine zusätzliche Komplikation dar (s. AL ALAMIN ET AL. 2020). Aus diesem Grund wird die folgende Regel festgelegt:

Regel: Schnittstellen für die Integration in bestehende IT-Architektur definieren und managen

Vor dem Hintergrund der skizzierten Problematiken ist die Definition und das Management von Schnittstellen für die Integration in die existierende IT-Infrastruktur von essenzieller Bedeutung. Angesichts des Mangels an standardisierten Schnittstellen und den Herausforderungen, die mit konfigurierbaren sowie individuellen Schnittstellen verbunden sind, ist es von fundamentaler Wichtigkeit, eine Zielarchitektur für die Integration der Low-Code-Plattform sowie der dazugehörigen Anwendungen zu definieren. Hierbei sind Leitlinien für die Gestaltung und Entscheidungsfindung der Integration

festzulegen und Schnittstellen zu spezifizieren (s. TIEMEYER 2020, S. 88). Bei den Schnittstellen lässt sich zwischen standardisierten, konfigurierbaren und individuellen Schnittstellen unterscheiden (siehe Kapitel 2.2.3). Strategische Schnittstellen sollten als Standards definiert werden, wobei bei der Gestaltung der Schnittstellen ein Fokus auf Modularität und Flexibilität zu legen ist, um Skalierbarkeit und Effizienz zu fördern (s. TIEMEYER 2020, S. 88). Die präzise Definition der Schnittstellen ist ausschlaggebend für eine nahtlose Kommunikation zwischen den verschiedenen Systemen (s. PRESSMAN U. MAXIM 2020, S. 175). Zur weiteren Verbesserung der Integration und Kommunikation kann es vorteilhaft sein, die notwendigen Schnittstellen für die Entwicklung als APIs zu implementieren (siehe Kapitel 2.2.3, LUO ET AL. 2021). Um die Interaktion zwischen den Systemen nachhaltig zu vereinfachen, sollte für sämtliche vorhandenen Schnittstellen ein effektives Management etabliert werden (OVEREEM ET AL. 2021).

Zusammenfassend lässt sich feststellen, dass die erfolgreiche Integration von Low-Code-Plattformen in bestehende IT-Infrastrukturen durch die sorgfältige Definition und das Management von Schnittstellen wesentlich gefördert wird, wobei die Berücksichtigung von Modularität, Flexibilität und der Einsatz von APIs als entscheidende Faktoren für die Optimierung der Interoperabilität und die Steigerung der Systemeffizienz hervorzuheben sind.

Qualität

Die Regel „Qualität“ umfasst auch das Management von Geschäftsprozesskontrollen. Der Fokus liegt dabei auf der Etablierung von Verfahren, welche die Qualität von mittels Low-Code entwickelten Anwendungen gewährleisten. Da Low-Code-Anwendung oft von weniger erfahrenen Entwickelnden erstellt werden, besteht eine erhöhte Fehleranfälligkeit (s. INDAMUTSA ET AL. 2021). Ein Hauptgrund hierfür sind oft unzureichende Testverfahren (s. BINZER U. WINKLER 2022; ELSHAN ET AL. 2023; IHIRWE ET AL. 2021; ROKIS U. KIRIKOVA 2022), wodurch das Risiko einer geringen Qualität der entwickelten Anwendungen erhöht ist (s. ELSHAN ET AL. 2023; HEUER ET AL. 2022; INDAMUTSA ET AL. 2021; LETHBRIDGE 2021). Typische Qualitätsmängel äußern sich in Form von Code-Redundanzen (s. ELSHAN ET AL. 2023), fehlender Dokumentation (s. ROKIS U. KIRIKOVA 2022) und dem weiteren Aufbau von Technischer Schuld (siehe Kapitel 2.1.3) (s. ELSHAN ET AL. 2023; LETHBRIDGE 2021). Folglich wird folgende Regel eingeführt:

Regel: Qualitätssicherungsprozess und Testing Framework für entwickelte Anwendungen einführen

Um potenzielle Fehler zu minimieren, sollte ein Prozess zur Qualitätskontrolle aller entwickelten Anwendungen vor deren Einsatz etabliert werden (s. ELSHAN ET AL. 2023). Hierbei müssen sowohl die funktionalen als auch nicht-funktionalen Anforderungen (siehe Kapitel 6.2.4) umfassend geprüft werden (s. ROKIS U. KIRIKOVA 2022). Im Rahmen des Qualitätssicherungsprozesses sollte festgelegt werden, wie die einzelnen Anforderungen überprüft werden und wer dafür zuständig ist. Zur Überprüfung von nicht-funktionalen Anforderungen eignet sich die Einführung eines Low-Code-Testing-

Frameworks (s. ROKIS U. KIRIKOVA 2022; KHORRAM ET AL. 2020). Das Testing-Framework ermöglicht eine effiziente und idealerweise automatisierte Testdurchführung (s. PRINZ ET AL. 2022; ROKIS U. KIRIKOVA 2022). Die idealerweise automatisierte Überprüfung der nicht-funktionalen Anforderungen hilft dabei, den Aufbau von technischer Schuld zu vermeiden (siehe Kapitel 2.1.3). Neben der automatisierten Testdurchführung sollten weitere Kontrollmechanismen zur Überprüfung der restlichen Anforderungen in den Qualitätssicherungsprozess integriert werden. Hierbei ist wichtig festzulegen, welche der zuvor definierten Rollen, für welche der Kontrollen verantwortlich ist. Je nach Anwendungstyp sollten diese Kontrollen von IT-Expert*innen durchgeführt werden (s. HEUER ET AL. 2022).

Zusammenfassend empfiehlt es sich, einen Qualitätssicherungsprozess mit möglichst automatisiertem Testing-Framework und Verantwortlichkeiten für alle weiteren Überprüfungen zu definieren, um den Aufbau von technischer Schuld zu vermeiden und die Qualität der Anwendungen zu gewährleisten.

Abbildung 6-13 fasst die Low-Code-Regeln in der Dimension „Planen“ zusammen. Innerhalb dieser Dimension konzentrieren sich die Regeln auf die Integration der Low-Code-Plattform in die IT-Architektur und die Gewährleistung der Qualität der Anwendungen.

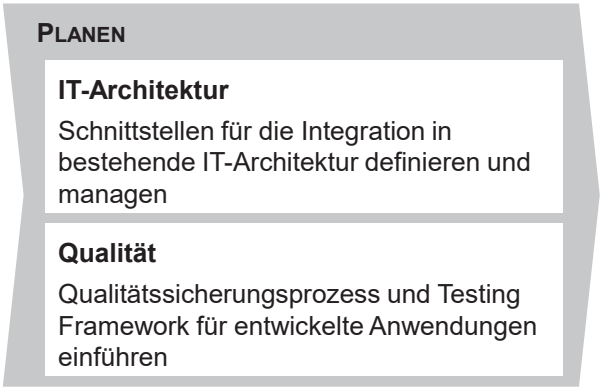

Abbildung 6-13: Low-Code-Regeln in der Dimension „Planen" (eigene Darstellung)

6.2.3 Aufbauen

In der Dimension „Aufbauen" liegt der Schwerpunkt auf der Definition und Entwicklung von Low-Code-Anwendung. Das Ziel besteht darin sicherzustellen, dass die Implementierung, Konfiguration und Anpassung der Anwendungen optimal mit den Geschäftsabläufen harmonieren und den angestrebten Mehrwert liefern. Innerhalb dieser Dimension konzentrieren sich die Low-Code-Regeln auf die Anforderungsdefinition und die Lösungserstellung der Low-Code-Anwendung.

Anforderungsdefinition

Bei der Regel „Anforderungsdefinition“ geht es darum festzuhalten, wie die Anforderungen für Low-Code-Anwendung definiert werden und wie mit Änderungen dieser Anforderungen umgegangen wird.

Bei der Anforderungsdefinition für Low-Code-Anwendung besteht das Risiko, dass die Anforderungen nicht ausreichend klar formuliert sind (s. ROKIS U. KIRIKOVA 2022). Besonders nicht-funktionale Anforderungen werden oft vernachlässigt. Benutzerfreundlichkeit und Skalierbarkeit, als Beispiele für nicht-funktionale Anforderungen, werden in der Regel erst von IT-Expert*innen während der Entwicklung berücksichtigt und definiert (s. ESCALONA ET AL. 2022). Dies führt bei der Low-Code-Entwicklung durch Nicht-IT-Expert*innen oft zu mangelnder Benutzerfreundlichkeit und einer schlechten Leistung der resultierenden Anwendungen (s. LUO ET AL. 2021; PRINZ ET AL. 2022). Abhängig von der Komplexität der Anforderungen und den geforderten Leistungsmerkmalen kann es zudem vorkommen, dass die technischen Möglichkeiten von Low-Code nicht ausreichen, um alle Anforderungen zu erfüllen (s. LUO ET AL. 2021). Zusätzlich besteht das Risiko, dass die definierten Anforderungen während der Entwicklung oder nach der ersten Einsatzphase der Anwendungen geändert werden (s. ROKIS U. KIRIKOVA 2022), wodurch Entscheidungen über die Priorisierung solcher Änderungen getroffen werden müssen (s. HEUER ET AL. 2022). Deshalb wird die nachstehende Regel definiert:

Regel: Anforderungsmanagementprozess und Kriterienkatalog für Anforderungen an die Low-Code-Anwendung aufsetzen

Die Etablierung eines klaren Prozesses zur Priorisierung und Implementierung von Änderungen (s. PRINZ ET AL. 2022) sowie die Einbindung eines Anforderungsmanagement-Tools (s. ROKIS U. KIRIKOVA 2022) zur Handhabung sich verändernder Anforderungen unterstützt das Management von IT-Änderungen. Dies ermöglicht eine gezielte und effiziente Reaktion auf dynamische Anforderungsänderungen während des Entwicklungsprozesses. Zusätzlich ist es für eine optimale Formulierung der Anforderungen notwendig, diese in ihrer Gesamtheit klar und umfassend zu definieren (s. PRINZ ET AL. 2022; ROKIS U. KIRIKOVA 2022). Dies schließt nicht nur eine ausführliche Beschreibung der funktionalen Anforderungen ein, sondern auch das präzise Erfassen nicht-funktionaler Anforderungen, insbesondere unter Berücksichtigung der Benutzerfreundlichkeit (s. ELSHAN ET AL. 2023). Um diese Anforderungen effektiv zu strukturieren, kann ein Kriterienkatalog auf Basis der formulierten Anforderungen entwickelt werden. Dieser Katalog ermöglicht eine systematische Überprüfung geplanter Anwendungen noch vor dem Beginn der Entwicklungsarbeit hinsichtlich ihrer Eignung für den Einsatz von Low-Code-Plattformen. Ein solcher Ansatz führt zu erhöhter Klarheit und minimiert potenzielle Fehlerquellen (s. ELSHAN ET AL. 2023).

Zusammenfassend ist ein effektives Anforderungsmanagement, das sowohl flexible Anpassungen an dynamische Entwicklungsbedingungen ermöglicht als auch funktionale sowie nicht-funktionale Anforderungen berücksichtigt, entscheidend für die Anforderungsdefinition von Low-Code-Anwendung.

Lösungserstellung

Bei der Lösungserstellung wird festgelegt, wie die zuvor definierten Anforderungen möglichst effizient und einheitlich umgesetzt werden können.

Im Rahmen des Low-Code-Entwicklungsprozesses, der die Gestaltung von Benutzeroberflächen sowie die Definition von Geschäftslogiken und Workflows einschließt (s. SAHAY ET AL. 2020), treten spezifische Schwierigkeiten auf. Ein Hauptproblem liegt in der limitierten Anpassungsfähigkeit von Designs und Layouts, die zu einer eingeschränkten Gestaltungsfreiheit führen können (s. LUO ET AL. 2021). Zudem können unterschiedliche Benutzeroberflächen (UIs) zu Inkonsistenzen in der Anwendungslandschaft führen, was die Nutzererfahrung beeinträchtigen kann (s. ELSHAN ET AL. 2023; ROKIS U. KIRIKOVA 2022). Bei der Definition von Geschäftslogiken und Workflows besteht das Risiko, dass sich die entwickelten Anwendungen stark unterscheiden, wenn mehr als eine Low-Code-Plattform im Einsatz ist (s. IHIRWE ET AL. 2021). Da derzeit wenig Standards und Coderichtlinien zur Low-Code-Programmierung existieren (s. IHIRWE ET AL. 2021), kann dies zu Mehrdeutigkeiten bei Terminologien in der Code Basis führen (s. INDAMUTSA ET AL. 2021). Daher empfiehlt es sich, unternehmensweite Standards zu etablieren (s. ROKIS U. KIRIKOVA 2022). Aufgrund dessen wird folgende Regel formuliert:

Regel: Standards für User Interface und entwickelte Workflows definieren und wiederverwertbare Komponenten nutzen

Die Einführung von Standards für die Benutzeroberfläche sowie für die Geschäftslogiken und Workflows unterstützt dabei, die Heterogenität der Anwendungen zu minimieren. Um grafischen Inkonsistenzen in der Anwendungslandschaft entgegenzuwirken, sollte die Entwicklung der Benutzeroberfläche nur unter Berücksichtigung von Designrichtlinien erfolgen (s. ELSHAN ET AL. 2023). Mithilfe von automatisierten Umwandlungen von Designelementen können die Designrichtlinien systemseitig festgelegt und eingehalten werden (s. ROKIS U. KIRIKOVA 2022). In Bezug auf die Code Basis spielen die Festlegung von klaren Kodierungsrichtlinien (s. ELSHAN ET AL. 2023) eine wesentliche Rolle, um bereits existierende Workflows und Designelemente wiederverwenden zu können (s. INDAMUTSA ET AL. 2021).

Zusammenfassend empfiehlt es sich, Standards für das User Interface sowie die entwickelten Geschäftslogiken und Workflows zu definieren, um wiederverwendbare Komponenten zu nutzen. Diese Maßnahmen tragen dazu bei, die Qualität zu steigern und die Effizienz der Lösungserstellung sicherzustellen.

Abbildung 6-14 fasst die Low-Code-Regeln in der Dimension „Aufbauen“ zusammen. Innerhalb dieser Dimension konzentrieren sich die Regeln auf die Anforderungsdefinition und die Lösungserstellung der Low-Code-Anwendungen.

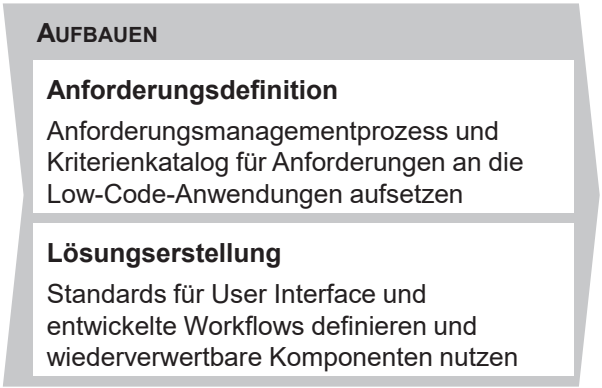

Abbildung 6-14: Low-Code-Regeln in der Dimension „Aufbauen" (eigene Darstellung)

6.2.4 Ausführen

In der Dimension „Ausführen" liegt der Fokus auf dem Betrieb und dem Support von Low-Code-Anwendung. Das Hauptziel besteht darin, sicherzustellen, dass diese Anwendungen reibungslos und ohne Unterbrechungen laufen und, dass schnell auf Serviceausfälle und Probleme reagiert werden kann. Innerhalb dieser Dimension konzentrieren sich die Regeln auf den Betrieb und den Support der Anwendungen.

Betrieb

Die Regel „Betrieb“ umfasst die Bereitstellung und das Verwalten der mit Low-Code erstellten Anwendungen sowie deren adaptive Wartung, um die Anwendung an unterschiedliche Betriebsumgebungen anzupassen (s. UMUDOVA 2019).

Die reibungslose Bereitstellung und der effiziente Betrieb von Low-Code-Anwendung sind essenziell und abhängig von den spezifischen Anforderungen und der bestehenden IT-Architektur eines Unternehmens (s. PRINZ ET AL. 2022). Der Bereitstellungsprozess umfasst die Zusammenstellung verschiedener Programmkomponenten, Daten und Bibliotheken sowie das Kompilieren und Verlinken dieser Komponenten zur Erstellung eines ausführbaren Systems (siehe Kapitel 2.1.3, SOMMERVILLE 2016, S. 731). Da sich die Konfiguration des Systems während der Entwicklung und Nutzung aufgrund von Änderungen in der IT-Architektur des Unternehmens fortlaufend ändern kann (s. SOMMERVILLE 2016, S. 731), ist eine ständige Wartung des Systems notwendig. Eine weitere Herausforderung beim Betrieb von Low-Code-Anwendung liegt in der Versionsverwaltung der Low-Code-Plattform (s. ROKIS U. KIRIKOVA 2022). Die Plattformen, auf denen Low-Code-Anwendung entwickelt werden, unterliegen in der Regel schnellen Weiterentwicklungen, wobei neue Hauptversionen eine Anpassung und Wartung der Low-Code-Anwendung erfordern, um ihre reibungslose Funktion sicherzustellen (s. LETHBRIDGE 2021). Neben der möglichen Anpassung bereits entwickelter Low-Code-Anwendung können neue Versionen von Low-Code-Plattformen auch Änderungen im Funktionalitätsumfang mit sich bringen. Daher ist es von entscheidender

Bedeutung, diese Änderungen zu erkennen und den neuen Funktionalitätsumfang bei der Entwicklung neuer Low-Code-Anwendung zu nutzen (s. HEUER ET AL. 2022). Insgesamt besteht die Herausforderung, den Betrieb der Low-Code-Anwendung sicherzustellen, dabei eine effiziente Wartung und Instandhaltung durchzuführen und gleichzeitig das System kontinuierlich zu optimieren (s. PRINZ ET AL. 2022). Aus diesem Anlass wird die folgende Regel bestimmt.

Regel: Zielbetriebsmodell und Bereitstellungsprozess festlegen und durch Versionskontrollen kontinuierlich optimieren

Um eine verlässliche Bereitstellung sicherzustellen, empfiehlt es sich, ein umfassendes Zielbetriebsmodell zu erstellen (s. PRINZ ET AL. 2022). Dieses Modell sollte nicht nur das reibungslose Zusammenspiel der unterschiedlichen Programmkomponenten, Daten und Bibliotheken innerhalb der entwickelten Low-Code-Anwendung sicherstellen, sondern auch deren Integration mit der restlichen Technologieinfrastruktur berücksichtigen und den spezifischen Anforderungen des Unternehmens gerecht werden. Viele Low-Code-Plattformen bieten Funktionen an, die beim Bereitstellungsprozess und bei der Durchführung der Wartungsarbeiten unterstützen (s. ROKIS U. KIRIKOVA 2022), was bei der Ausgestaltung des Zielbetriebsmodells genutzt werden sollte. Darüber hinaus sollte der Prozess zur Bereitstellung neuer Anwendungen klar definiert sein, um unter anderem sicherzustellen, dass vor jeder Bereitstellung neuer Anwendungen Backups angelegt werden (s. INDAMUTSA ET AL. 2021). Ebenso wichtig ist die Ermöglichung einer effektiven Versionskontrolle. Die höhere Transparenz neuer Versionen verbessert nicht nur die Erfahrung der Entwickelnden, sondern steigert auch die Lesbarkeit, Benutzerfreundlichkeit und Skalierbarkeit der Anwendungen (s. LETHBRIDGE 2021).

Zusammenfassend erfordert die erfolgreiche Bereitstellung und Wartung von Low-Code-Anwendung ein sorgfältig konzipiertes Zielbetriebsmodell, einen klar definierten Bereitstellungsprozess und eine effektive Versionskontrolle, um die Anwendungen kontinuierlich an sich ändernde Anforderungen und technologische Entwicklungen anzupassen.

Support

Die Regel „Support“ beinhaltet die Gewährleistung einer schnellen und effektiven Bearbeitung von Benutzeranforderungen und die Lösung aller Arten von Störungen, sowie die Bereitstellung zügiger Lösungen, um eine erneute Störung zu vermeiden.

Die Entwicklung und Instandhaltung von Low-Code-Anwendung sind ohne adäquate Unterstützung besonders herausfordernd. Wartungsprozesse können ohne die richtigen Ressourcen und Unterstützungssysteme zeitintensiv, schwer zu pflegen und anfällig für Fehler sein (s. INDAMUTSA ET AL. 2021). Insbesondere der Support nach Bereitstellung der Anwendungen stellt eine große Herausforderung dar (s. LUO ET AL. 2021; PRINZ ET AL. 2022). Die Notwendigkeit, auf nachträglich auftretende Anforderungen effektiv zu reagieren, ohne dabei unerwünschte Nebeneffekte zu erzeugen, erfordert zusätzliche Kapazitäten und eine vorausschauende Planung (s. DI RUSCIO ET AL.

2022). In der Praxis der Softwarewartung lassen sich verschiedene Wartungsarten unterscheiden, darunter korrektive Wartung zur Behebung von Fehlern, perfektionierende Wartung zur Verbesserung der Funktionalität oder Performance und präventive Wartung zur Vermeidung zukünftiger Probleme (s. UMUDOVA 2019). Folglich wird die folgende Regel festgelegt:

Regel: Verantwortlichkeiten für Wartungsaufgaben festlegen und weitergehenden Support definieren

Um die Effektivität des Supports und die Zufriedenheit der Anwender*innen zu steigern, ist es essenziell, klare Verantwortlichkeiten für die Wartungsaufgaben jeder Anwendung zu definieren und umfassenden Support festzulegen. HEUER ET AL. empfehlen, präzise zu bestimmen, wer für die Betreuung und Unterstützung spezifischer Anwendungen verantwortlich ist. Dies beinhaltet die Zuordnung von Fachkräften, die über die erforderlichen Kenntnisse und Fähigkeiten verfügen, um die jeweilige Anwendung optimal zu unterstützen. Die Differenzierung nach der Art der Anwendung und den damit verbundenen spezifischen Anforderungen ist dabei von großer Bedeutung, um sicherzustellen, dass jede Anwendung die benötigte Aufmerksamkeit und Fachexpertise erhält (s. HEUER ET AL. 2022). Zusätzlich zur Festlegung von Verantwortlichkeiten ist die Implementierung eines robusten Support-Systems ratsam, wodurch nicht nur korrektiv Fehler behoben werden, sondern auch Raum für Verbesserungsvorschläge und präventive Maßnahmen geboten werden kann (vgl. Kapitel 2.1.3).

Zusammenfassend unterstreicht die Komplexität der Entwicklung und Wartung von Low-Code-Anwendung die Notwendigkeit einer umfassenden Unterstützungsstruktur, die klare Verantwortlichkeiten, spezialisiertes Fachwissen und ein robustes Support-System umfasst, um zeit- und ressourcenaufwendige Prozesse zu optimieren, die Effektivität des Supports zu maximieren und die Zufriedenheit der Anwender*innen dauerhaft zu gewährleisten.

Abbildung 6-15 fasst die Low-Code-Regeln in der Dimension „Ausführen“ zusammen. Innerhalb dieser Dimension konzentrieren sich die Regeln auf den Betrieb und den Support der Anwendungen.

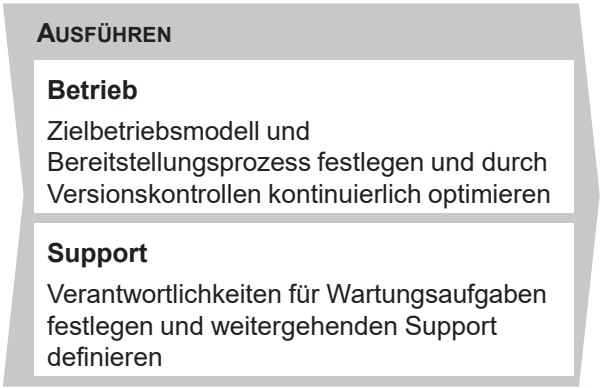

Abbildung 6-15: Low-Code-Regeln in der Dimension „Ausführen" (eigene Darstellung)

6.2.5 Überwachen

Der Bereich „Überwachen" konzentriert sich auf das interne Kontrollsystem und die Erfüllung von externen Compliance-Anforderungen. Ziel ist es zu gewährleisten, dass die Überprüfung der Anwendungen hinsichtlich der zu erfüllenden Anforderungen möglich ist und entsprechende Maßnahmen zur Compliance-Einhaltung identifiziert werden können. Innerhalb dieser Dimension konzentrieren sich die Regeln auf die Überwachung und Verwaltung der Daten sowie die Sicherstellung des Wissensmanagements und der IT-Sicherheit.

Daten

Die Regel „Daten“ umfasst das Management der Daten im gesamten Lebenszyklus aller mit Low-Code erstellten Anwendungen von der Erstellung über die Bereitstellung und Wartung bis hin zur Archivierung.

Eine bedeutende Herausforderung besteht darin, die Compliance-Vorschriften im Zusammenhang mit Informationssicherheitsmanagementsystemen (ISMS) (s. PRINZ ET AL. 2022) und den Datenschutzbestimmungen (s. ELSHAN ET AL. 2023; INDAMUTSA ET AL. 2021) einzuhalten. Die einfache Entwicklung von Software mithilfe von Low-Code-Plattformen ermöglicht Entwickelnden einen unkomplizierten Zugriff auf verschiedene Arten von Daten (s. HEUER ET AL. 2022; ROKIS U. KIRIKOVA 2022). Die Gewährleistung der Sicherheit, Privatsphäre und Integrität von Diensten und Daten aus unterschiedlichen Quellen stellt daher eine große Herausforderung dar (s. INDAMUTSA ET AL. 2021). Infolgedessen wird die nachstehende Regel definiert:

Regel: Regeln für Umgang mit Daten festlegen und durch Zugriffsmanagement verwalten

Um diesen Herausforderungen zu begegnen, sollten Maßnahmen ergriffen werden, um Benutzer*innen bei der Einhaltung der Compliance-Anforderungen zu unterstützen. Dies kann durch die Bereitstellung von Richtlinien, Materialien und konformen

Schnittstellen erfolgen (s. HEUER ET AL. 2022). Zusätzlich ist es von entscheidender Bedeutung, die Sicherheit, Privatsphäre und Integrität von Diensten und Daten, die aus verschiedenen Quellen stammen, zu gewährleisten. Dies erfordert eine gründliche und umfassende Datenverwaltung, die sicherstellt, dass alle Daten durchgehend geschützt sind (s. PRINZ ET AL. 2022). Eine solide Datenverwaltung wird durch die Implementierung einer umfassenden Zugangskontrollverwaltung ermöglicht, welche den Zugriff auf Daten und Dienste streng reguliert (s. HUSSAIN ET AL. 2020). Dies stellt sicher, dass nur autorisierte Personen Zugriff auf sensible Informationen erhalten und dass die festgelegten Richtlinien für den Umgang mit Daten konsequent eingehalten werden (s. PRINZ ET AL. 2022). Die Festlegung klarer Regeln für den Umgang mit Daten und die Implementierung eines effektiven Zugriffsmanagements sind somit zentral, um den Anforderungen im Zusammenhang mit Compliance und Datenschutz gerecht zu werden und um eine sichere Nutzung der Low-Code-Plattformen zu ermöglichen.

Zusammengefasst erfordert das effektive Datenmanagement in der Low-Code-Entwicklung eine strategische Kombination aus technischen Maßnahmen, organisatorischen Richtlinien und einer Kultur der Sicherheit und Compliance. Durch die Einrichtung strenger Zugriffsmanagement-Systeme und die Bereitstellung von Unterstützungsmaßnahmen für die Einhaltung von Datenschutz- und Compliance-Vorgaben können Unternehmen die Herausforderungen meistern und eine sichere, zuverlässige und regelkonforme Nutzung von Low-Code-Anwendung gewährleisten.

Wissen

Die Regel „Wissen" befasst sich mit der Aufrechterhaltung der Verfügbarkeit von Wissen über die mit Low-Code erstellten Anwendungen.

Die Befähigung von Mitarbeiter*innen aus den Fachabteilungen zur eigenständigen Entwicklung von Lösungen mittels Low-Code-Anwendung eröffnet zwar neue Möglichkeiten, jedoch geht dies auch mit einigen Herausforderungen einher. Insbesondere ist die Schatten-IT ein wichtiger Gesichtspunkt, der beachtet werden muss (s. ELSHAN ET AL. 2023). Eine weitere Herausforderung ergibt sich aus dem Mangel an Transparenz über die entwickelten Lösungen (s. HEUER ET AL. 2022). Wenn verschiedene Abteilungen innerhalb eines Unternehmens eigenständig Low-Code-Anwendung entwickeln, kann die Gesamtübersicht über diese Anwendungen verloren gehen. Dies erschwert die effiziente Ressourcennutzung, die Koordination von IT-Ressourcen und die Identifizierung von Duplikaten oder ähnlichen Lösungen. Darüber hinaus stellt die mangelnde Dokumentation eine weitere Herausforderung dar, sowohl hinsichtlich dessen, was die Entwickelnden in ihren Anwendungen erstellen, als auch in Bezug auf die Dokumentation, die in den Code integriert werden kann (s. LETHBRIDGE 2021). Diese unzureichende Dokumentation erschwert nicht nur die Wartung und Weiterentwicklung der Anwendungen, sondern beeinträchtigt auch die Möglichkeit, Workflows und Funktionen aus bestehenden Anwendungen effektiv wiederzuverwenden (s. INDAMUTSA ET AL. 2021). Daher wird folgende Regel festgelegt:

Regel: Übersicht mit Dokumentation zu allen entwickelten Anwendungen erstellen

Die Bewältigung dieser Herausforderungen kann durch die Etablierung eines umfassenden Wissensmanagements in Bezug auf sämtliche Anwendungen sowie die Sicherstellung eines umfassenden Überblicks über diese Anwendungen erreicht werden (s. PRINZ ET AL. 2022). Darüber hinaus ist es von entscheidender Bedeutung, die Funktionen und wiederverwendbaren Workflows dieser Anwendungen ausreichend zu dokumentieren (s. HEUER ET AL. 2022; INDAMUTSA ET AL. 2021; PRINZ ET AL. 2022; ROKIS U. KIRIKOVA 2022). Eine umfassende Dokumentation der Anwendungen ist von entscheidender Bedeutung, um ihre langfristige Nutzbarkeit und Erweiterbarkeit sicherzustellen. Dies trägt nicht nur dazu bei, die Schatten-IT unter Kontrolle zu halten, sondern gewährleistet auch eine klare Transparenz über die entwickelten Lösungen und steigert die Effizienz bei der Anwendungsentwicklung (s. INDAMUTSA ET AL. 2021). Viele Low-Code-Plattformen bieten bereits integrierte Möglichkeiten zur Überwachung und Dokumentation von Anwendungen. Es ist ratsam, diese Werkzeuge zu nutzen, um das Wissen über die erstellten Anwendungen effektiv zu verwalten und zu optimieren (s. ELSHAN ET AL. 2023).

Zusammenfassend sind ein umfassendes Wissensmanagement, eine gründliche Dokumentation der Anwendungen sowie die Nutzung von integrierten Überwachungs- und Dokumentationswerkzeugen der Low-Code-Plattformen erforderlich, um langfristige Nutzbarkeit und effiziente Ressourcennutzung zu gewährleisten.

IT-Sicherheit

Die Regel „IT-Sicherheit“ befasst sich mit dem Schutz der Unternehmensinformationen, die bei der Entwicklung der Low-Code-Anwendung verwendet werden. Darunter fallen die Einrichtung und Pflege von Informationssicherheitsrollen und Zugangsrechten sowie die Durchführung von Sicherheitsüberwachungsmaßnahmen.

Eine Herausforderung in der Low-Code-Entwicklung besteht darin, dass Entwickelnde oft nicht über das notwendige Wissen bezüglich erforderlicher Sicherheitspraktiken verfügen (s. IHIRWE ET AL. 2021). Besonders bei der Integration externer Dienste über APIs (vgl. Kapitel 2.2.3) ist die Implementierung von Sicherheitsmaßnahmen entscheidend, um ein angemessenes Sicherheitsniveau zu gewährleisten (s. INDAMUTSA ET AL. 2021), da sie Zugang zu wertvollen und geschützten Daten und Ressourcen bieten (s. MATHIJSSEN ET AL. 2020). Ähnlich müssen auch die Schnittstellen zwischen den verschiedenen Systemen überwacht werden, um die Sicherheit der Datenübertragung und die Integration zwischen verschiedenen Systemen zu sichern (s. HEUER ET AL. 2022). Obwohl Low-Code-Plattformen modernen Sicherheitsstandards entsprechen und Entwickelnde in den meisten Sicherheitspraktiken unterstützen, müssen fortgeschrittene Sicherheitspraktiken von den Entwickelnden selbst implementiert werden (s. OVEREEM ET AL. 2021). Dies betont die Notwendigkeit, dass Low-Code-Entwickelnde nicht ausschließlich auf die Sicherheitsmaßnahmen der Plattformanbieter vertrauen, sondern eigene Sicherheitsprotokolle und -praktiken entwickeln und einführen müssen, um ein umfassendes Sicherheitsniveau sicherzustellen. Infolgedessen wird die nachstehende Regel eingeführt:

Regel: Organisatorische und technische Sicherheitspraktiken entwickeln und umsetzen

Neben der Sicherstellung der Betriebssicherheit (siehe Kapitel 2.3.5) durch organisatorische Maßnahmen, wie das Schärfen des Bewusstseins für Sicherheitsprobleme, sind weitere technische Maßnahmen notwendig, um die Infrastruktur- und Anwendungssicherheit (siehe Kapitel 2.3.5) zu stärken. Um Zielnutzer zu unterstützen, die möglicherweise nicht über das erforderliche Wissen bezüglich der anzuwendenden technischen Sicherheitspraktiken verfügen, sind geeignete technische Abstraktionen und Automatisierungstechniken erforderlich, die grundlegende Sicherheitspraktiken umsetzen (s. IHIRWE ET AL. 2021). APIs kommen im Kontext der IT-Sicherheit eine Schlüsselrolle zu (s. REDDY U. RUDRA 2021) und sollten daher unter besonderer Überwachung stehen. MATHIJSSEN ET AL. empfehlen umfangreiche technische Maßnahmen, um Sicherheitsrisiken zu verhindern. Dies umfasst unter anderem die Implementierung geeigneter Authentifizierung und Autorisierung, Maßnahmen gegen Bedrohungserkennung und -schutz sowie das Verschlüsseln sensibler Daten (s. MATHIJSSEN ET AL. 2020).

Zusammenfassend sollten sowohl organisatorische als auch technische Maßnahmen zur IT-Sicherheit beitragen. Dabei sollte ein spezieller Fokus auf die Implementierung von technischen Abstraktionen und Automatisierungstechniken gelegt werden, um insbesondere den potenziellen Sicherheitslücken von APIs entgegenzuwirken.

Abbildung 6-16 fasst die Low-Code-Regeln in der Dimension „Überwachen“ zusammen. Innerhalb dieser Dimension konzentrieren sich die Regeln auf die Überwachung und Verwaltung der Daten sowie die Sicherstellung des Wissensmanagements und der IT-Sicherheit.

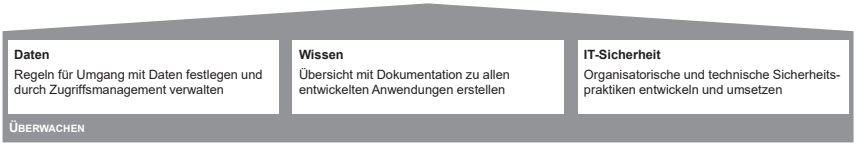

Abbildung 6-16: Low-Code-Regeln in der Dimension „Überwachen" (eigene Darstellung)

6.3 Reflexion und Zusammenfassung der Ergebnisse

In den vorangegangenen Kapiteln wurden Regeln für Low-Code-Anwendung in produzierenden Unternehmen entwickelt. Durch die systematische Literaturrecherche in Kapitel 3.2.1 konnten erste wichtige wissenschaftliche Einsichten zu Low-Code-Regelwerken und einer entsprechenden Governance erlangt werden. Die Ergebnisse zeigten allerdings, dass eine direkte Übertragung dieser Erkenntnisse auf den spezifischen Anwendungsbereich nur begrenzt möglich ist. Aus diesem Grund wurden in diesem Kapitel Regeln für den Einsatz von Low-Code entwickelt und in einem strukturierten Regelwerk festgehalten.

Die Ausarbeitung der Regeln basierte auf einer Inhaltsanalyse nach MAYRING (siehe Kapitel 4.2.5). Hierzu wurde das COBIT-Kernmodell genutzt, um Strukturierungsdimensionen und Regeln festzulegen (siehe Kapitel 6.1.1). Um die Untersuchung auf den Low-Code-Kontext zu spezifizieren, wurden 17 Quellen in Bezug auf Risiken, Herausforderungen und Regeln von Low-Code analysiert und für Low-Code-Regeln relevante Textstellen markiert. Die markierten Textstellen wurden bewertet und den entsprechenden Low-Code-Regeln zugeordnet (siehe Kapitel 6.1.2). Durch diese Vorgehensweise konnten insgesamt elf Low-Code-Regeln abgeleitet werden, die durch die fünf Dimensionen „Vorgeben“, „Planen“, „Aufbauen“, „Ausführen“ und „Überwachen“ strukturiert werden (siehe Kapitel 6.2). Das Regelwerk wird in Abbildung 6-17 praxisgerecht visualisiert, um eine umfassende Übersicht aller relevanten Aspekte zu bieten, die bei der Implementierung und dem Betrieb von Low-Code-Anwendung beachtet werden müssen.

Abbildung 6-17: Regeln für Low-Code-Anwendungen (eigene Darstellung)

Die Regeln bilden eine erste Orientierungshilfe für Unternehmen, die sich mit der Implementierung und dem Betrieb von Low-Code-Anwendungen auseinandersetzen möchten. Diese Ausarbeitung adressiert die in Kapitel 3.2 identifizierte Forschungslücke und beantwortet die zweite spezifische und richtungsweisende Forschungsfrage:

Wie können Regeln für Low-Code innerhalb eines produzierenden Unternehmens beschrieben werden?

Eine kritische Reflexion des entwickelten Regelwerks zeigt, dass es eine solide Grundlage für die praktische Anwendung bietet, jedoch auch einige Einschränkungen aufweist. Einerseits ermöglicht die Strukturierung nach COBIT-Dimensionen eine systematische Herangehensweise und erleichtert die Implementierung und Überwachung der Low-Code-Anwendungen. Andererseits könnten spezifische Unternehmensanforderungen oder branchenspezifische Besonderheiten zusätzliche Anpassungen der Regeln erfordern. Ein weiterer wichtiger Aspekt sind die kontinuierliche Anpassung

und Aktualisierung der Regeln, um mit den sich schnell entwickelnden Technologien und Marktanforderungen Schritt zu halten. Dies erfordert regelmäßige Überprüfungen und gegebenenfalls Anpassungen des Regelwerks.

Im folgenden Kapitel soll nun untersucht werden, welche Abhängigkeiten zwischen den in diesem Kapitel identifizierten Low-Code-Regeln und den abgeleiteten Low-Code-Anwendungsfalltypen aus Kapitel 5 bestehen. Dies wird dazu beitragen, die Anwendbarkeit und Relevanz der Regeln weiter zu validieren und möglicherweise notwendige Anpassungen zu identifizieren.

7 Wirkung von Anwendungsfällen auf Regeln

Das folgende Kapitel zielt darauf ab, die dritte untergeordnete Forschungsfrage zu beantworten:

Wie können aus den Low-Code-Anwendungsfällen Wirkbeziehungen zu den Low-Code-Regeln erklärt werden?

In Kapitel 5 wurden die verschiedenen Typen von Low-Code-Anwendungsfällen definiert und voneinander abgegrenzt. Kapitel 6 konzentrierte sich auf die Erarbeitung eines Low-Code Regelwerks, das bei der Einführung und dem Betrieb von Low-Code berücksichtigt werden muss. Das Ziel von Kapitel 7 ist es, die Abhängigkeiten zwischen den Low-Code-Anwendungsfalltypen und den Low-Code-Regeln zu identifizieren, um im darauffolgende#n Kapitel geeignete Gestaltungsmaßnahmen zur Einhaltung dieser Regeln für jeden Anwendungsfalltyp festzulegen. Zu diesem Zweck werden die Beschreibungsmodelle für Low-Code-Anwendungsfälle und Regeln in diesem Kapitel integriert. Es wird analysiert, welchen Einfluss die Merkmale und Ausprägungen der drei Anwendungsfalltypen auf die elf Regeln haben (siehe Abbildung 7-1).

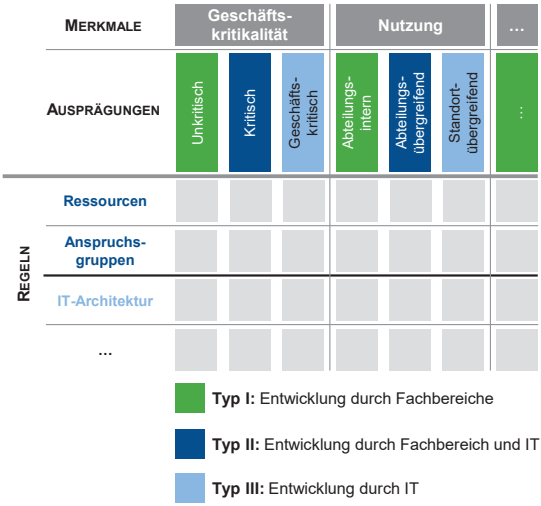

Abbildung 7-1: Struktur des Wirkmodells (eigene Darstellung)

Kapitel 7.1 erläutert die Methodik zur Untersuchung der Wirkbeziehungen, einschließlich der Auswahl geeigneter Datenquellen und einer systematischen Bewertungsmethodik. Angesichts der begrenzten Literatur aus der systematischen Literaturrecherche zu Low-Code-Risiken und Regeln (siehe Kapitel 3) werden zusätzlich Expert*innen-interviews durchgeführt, um die Wirkbeziehungen herzuleiten. In den Abschnitten 7.2 bis 7.6 werden dann die spezifischen Wirkbeziehungen für die verschiedenen Dimensionen des Regelwerks („Vorgeben“, „Planen“, „Aufbauen“, „Ausführen“ und „Überwachen“) untersucht. Dabei wird im Detail erläutert, welchen Einfluss die Merkmale und

Ausprägungen der Anwendungsfalltypen auf die einzelnen Regeln des Regelwerks haben. Eine Wirkbeziehung ist immer dann vorhanden, wenn aus dem Merkmal oder der Ausprägung des Anwendungsfalltyps eine Maßnahme zum Einhalten der Regel abgeleitet werden kann. Das Kapitel schließt mit einer Reflexion und Zusammenfassung der Ergebnisse, die als Basis für das abschließende Gestaltungsmodell in Kapitel 8 dienen.

7.1 Vorgehensweise zur Erklärung der Wirkbeziehungen

Die Notwendigkeit, die beiden zuvor beschriebenen Modelle in einem Modell zusammenzuführen und dessen Wirkbeziehungen zu erklären, ergibt sich direkt aus den Zielen dieser Dissertation.

Ziel dieser Dissertation ist die Entwicklung eines Regelwerks für Low-Code-Anwendungen, das spezifische Regeln und Maßnahmen für produzierende Unternehmen umfasst. Um aus den spezifischen Regeln aus Kapitel 6 anwendungsspezifische Maßnahmen ableiten zu können, werden in diesem Kapitel die Wirkzusammenhänge zwischen den Low-Code-Anwendungsfalltypen (siehe Kapitel 5) und Regeln (siehe Kapitel 6) untersucht. Ein Wirkzusammenhang ist dann vorhanden, wenn anhand des Merkmals oder der Ausprägung des Anwendungsfalltypen eine Maßnahme zum Einhalten der Regel abgeleitet werden kann. Lässt sich eine Maßnahme anhand des Merkmals ableiten, handelt es sich um eine typenunabhängige Maßnahme. Lässt sich eine Maßnahme anhand einer Merkmalsausprägung ableiten, handelt es sich um eine typenspezifische Maßnahme für den Anwendungsfalltyp mit der entsprechenden Ausprägung.

Obwohl es bei den Anwendungsfalltypen theoretisch möglich wäre, sich nur auf die Merkmale der Anwendungsfalltypen zu konzentrieren, zeigt sich, dass ein solcher Ansatz nicht den methodischen Mehrwert liefert, der angestrebt wird. Stattdessen sollen die Ergebnisse dieser Untersuchung eine Basis für die Erarbeitung von Maßnahmen bieten, die in der Unternehmenspraxis je nach Typ des Anwendungsfalls variieren und spezifisch angewendet werden. Daher werden bei der Analyse der Anwendungsfalltypen sowohl die Merkmalsebene für typenunabhängigen Maßnahmen als auch die Ausprägungsebene für die Ableitung typenspezifischer Maßnahmen betrachtet. Dies erfolgt für jede Regel in den fünf unterschiedlichen Dimensionen.

Aufgrund der geringen Anzahl an vorhandener Literatur zu Low-Code-Anwendungsfällen und Regeln (siehe Kapitel 3.1.1 und 3.2.1) verzichtet diese Dissertation auf eine quantitative Untersuchung dieser Zusammenhänge. Stattdessen werden die Wirkzusammenhänge sachlogisch und basierend auf Literaturzitaten hergeleitet oder durch Expert*inneninterviews (siehe A.5 bis A.11) identifiziert.

Die Identifizierung von Wirkzusammenhängen dient dazu, Handlungsalternativen einzugrenzen und den Einsatz des Regelwerks zwischen diesen Alternativen zu erleichtern. Diese Aufgabe kann durch eine intuitive und qualitative Bestimmung der Intensität der Beziehungen zwischen den betrachteten Objekten bewältigt werden (s. Probst u. Gomez 1991, S. 13). Das Verfahren zur Ermittlung dieser Zusammenhänge orientiert

sich an den Ausführungen von PROBST U. GOMEZ. Dabei kommt ein qualitatives Bewertungsschema zum Einsatz, das drei aufsteigende Intensitätsstufen umfasst (siehe Tabelle 18).

Tabelle 18: Qualitatives Bewertungsschema der Intensitätsstufen (eigene Darstellung)

Darstellung	Intensität	Beschreibung
	Keine oder eine äußerst geringe Intensität	Anhand des Merkmals oder der Ausprägung des Anwendungsfalltypen kann keine Maßnahme zum Einhalten der Regel abgeleitet werden. Die Einhaltung der Regel hat keinen Einfluss auf das Merkmal oder die Ausprägung des Anwendungsfalltypen.
+	Normale Intensität	Anhand des Merkmals des Anwendungsfalltypen kann eine Maßnahme zum Einhalten der Regel abgeleitet werden. Es handelt sich um Maßnahmen, die für alle Anwendungsfalltypen gültig sind.
++	Starke Intensität	Anhand der Ausprägung des Anwendungsfalltypen kann eine Maßnahme zum Einhalten der Regel abgeleitet werden. Es handelt sich um Maßnahmen, die nur für spezifische Anwendungsfalltypen gültig sind.

Diese Stufen lehnen sich an die Empfehlungen von PROBST U. GOMEZ. Sofern eine Intensität festgestellt wird (sei es + oder ++), können aus einer Regel Maßnahmen abgeleitet werden, die je nach Intensitätsgrad entweder allen drei Anwendungsfalltypen oder einem spezifischen Anwendungsfalltyp zugeordnet werden können.

Die Analyse der Wirkzusammenhänge wird methodisch durchgeführt und stützt sich auf die praktische Erfahrung der Autorin, auf umfassende Literaturrecherchen sowie auf Expert*inneninterviews (siehe A.5 bis A.11). Tabelle 19 zeigt einen Überblick über die Interviewpartner*innen. Die erzielten Ergebnisse werden textlich dargelegt und zusätzlich durch Matrizen visualisiert. Diese Visualisierungen zielen darauf ab, die Ergebnisse nicht nur transparent und deutlich darzustellen, sondern auch die erläuterten Inhalte kompakt zu aggregieren.

Tabelle 19: Übersicht der Interviewpartner (eigene Darstellung)

Lfd. Nr.	Anwendungsfalltyp	Mitarbeiter	Branche	Anhang
I1	III	< 400	Produzierendes Unternehmen	A.5
I2	II	< 2.000	IT und Services	A.6
I3	II	< 300	IT und Services	A.7
I4	I	< 25.000	Produzierendes Unternehmen	A.8
I5	III	< 100	IT und Services	A.9
I6	III	< 100	IT und Services	A.10
I7	III	< 100	IT und Services	A.11

Jede erstellte Matrix untersucht eine Dimension des Low-Code-Regelwerks (siehe Kapitel 6.2). Die Zeilen der Matrix repräsentieren die zu der Dimension gehörenden Regeln, während die Spalten der Matrix die zu dem Merkmal gehörenden Ausprägungen der Anwendungsfalltypen entsprechen (siehe Kapitel 5.2). Demnach wird für jede Dimension des Low-Code-Regelwerks eine eigene Matrix erstellt, wobei die Anzahl an Spalten der Anzahl aller Merkmalsausprägungen der Anwendungsfalltypen entspricht.

Diese methodische Herangehensweise zur Erfassung und Darstellung der Zusammenhänge sowie zur Darstellung der notwendigen Maßnahmen wird in Abbildung 7-1 veranschaulicht. Im nachfolgenden Kapitel wird diese Methode auf alle Dimensionen angewendet (siehe Kapitel 7.2).

In der Analyse werden ausschließlich die Wirkbeziehungen berücksichtigt, die entweder eine normale (+) oder starke (++) Intensität aufzeigen. Nichtexistierende Wirkbeziehungen werden nicht beschrieben. Diese Herangehensweise ermöglicht eine fokussierte und klare Darstellung der relevanten Zusammenhänge (siehe Tabelle 18).

7.2 Typenspezifische Wirkbeziehungen: Vorgeben

In diesem Abschnitt werden die Zusammenhänge zwischen den Anwendungsfalltypen und den Regeln in der Dimension „Vorgehen“ detailliert erörtert. Es wird aufgezeigt, wie aus jeder Regel Maßnahmen resultieren, die entweder für das Merkmal also für alle Anwendungsfalltypen (Normale Intensität) oder für bestimmte Merkmalsausprägungen also für spezifische Anwendungsfalltypen (Starke Intensität) gelten. Eine visuelle Zusammenfassung dieser zu untersuchenden Wirkbeziehungen, bezogen auf die Dimension „Vorgeben“, findet sich in Abbildung 7-2.

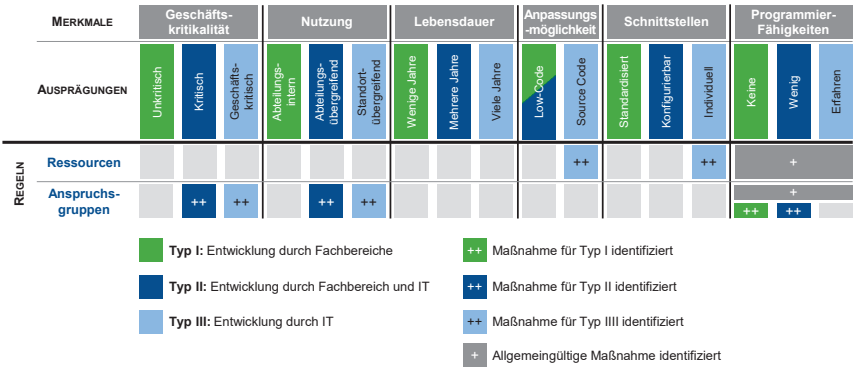

Abbildung 7-2: Wirkzusammenhänge in der Dimension „Vorgeben" (eigene Darstellung)

Ressourcen

Die Regel „Ressourcen“ gibt vor, dass ein Rollenmodell mit notwendigen Kapazitäten und Schulungen definiert werden soll. Die Regel sorgt dafür, dass die Low-Code-

Entwickelnden ausreichend geschult sind, um die geplanten Anwendungen zu entwickeln und dafür auch ausreichend Kapazität zur Verfügung gestellt bekommen.

Sobald die Low-Code-Plattform die Funktionalität anbietet, dass der Source Code angepasst werden kann, sollte es „Spezial Schulungen“ (siehe A.6) für die Entwickelnden geben, die den Source Code verändern dürfen. Außerdem sollten „nur dedizierte Leute den Source Code verändern können“ (siehe A.5), sodass das Rollenmodell angepasst werden muss. Insgesamt handelt es sich bei der Anpassungsfähigkeit durch den Source Code um eine starke Intensität, da sich bei diesem Fall spezifische Maßnahmen ableiten können. Eine starke Wirkbeziehung besteht zusätzlich bei den individuellen Schnittstellen. Traditionell werden die Aktivitäten innerhalb des Schnittstellen-Managements von technischem Personal durchgeführt. Die Stärke von Low-Code Development Plattformen besteht jedoch darin, dass Aktivitäten, die früher von erfahrenen Entwicklern durchgeführt wurden, nun von Citizen Developern ausgeführt werden können (s. Overeem et al. 2021). Dennoch muss „beim Erstellen neuer Schnittstellen teilweise sehr stark individualisiert werden“ (siehe A.7). Hier gibt es also besonderen Bedarf von Schulungsmaterial für das Erstellen von individuellen Schnittstellen.

Bei der Low-Code-Entwicklung ist es möglich, nicht nur das Schnittstellenmanagement, sondern die gesamte Softwareentwicklung mit keinen oder nur geringen Programmierfähigkeiten durchzuführen. Daher sind für die Entwickler*innen mit keinen oder geringen Programmierfähigkeiten spezifische Schulungen notwendig (s. Prinz et al. 2022; Elshan et al. 2023). Allerdings kann man festhalten, dass auch erfahrene Programmierer*innen Schulungen benötigen, da „erfahrene Programmierer sich viel zu stark auf ihre Programmierfähigkeiten verlassen und die Vorteile von Low-Code teilweise gar nicht nutzen“ (siehe A.7). Demnach sollten die Schulungen zwar abhängig von den Programmierfähigkeiten angepasst werden und „viele Freiräume bieten, sich das Wissen selbst anzueignen“ (siehe A.8), das Schulungskonzept muss aber für alle Programmierkenntnisse aufgebaut werden. Folglich kann man auch hier von einer normalen Intensität der Wirkbeziehung sprechen.

Anspruchsgruppen

Die Regel für „Anspruchsgruppen“ lautet: „Center of Excellence zum Erfahrungsaustausch und Entscheidungen treffen einführen“. Durch das Center of Excellence sollen die Low-Code-Entwickelnden zusammengebracht werden, um die Anwendungsentwicklung durch den Austausch stetig zu verbessern.

„Bei unkritischen Anwendungen geht man nach dem Standard vor, und nimmt das, was die Organisation vorgegeben hat“ (siehe A.7). Der Anspruch, sich regelmäßig auszutauschen und stetig zu verbessern, ist demnach vernachlässigbar. Es besteht keine Wirkbeziehung zwischen unkritischen Anwendungen und Anspruchsgruppen, wohingegen „bei kritischen oder geschäftskritischen Anwendungen ein eigenes Projektteam geschaffen“ (siehe A.6) wird. Die Schaffung eines eigenen Projektteams zeigt, wie wichtig es bei diesen Anwendungen ist, die Anspruchsgruppen zusammenzubringen und sich regelmäßig auszutauschen. Man kann also eine starke Wirkbeziehung zwischen kritischen und geschäftskritischen Anwendungen und den

Anspruchsgruppen annehmen. Gleiches zeigt sich bei der Nutzung der Anwendungen. „Wenn etwas abteilungsintern entwickelt wird, ist die Person, die die Anwendung entwickelt, sehr wahrscheinlich auch die Person, die die Anwendung nutzt“ (siehe A.9). „Wenn etwas standortübergreifend entwickelt wird, hat das Projektteam eine andere Struktur“ (siehe A.9). Bei „abteilungsübergreifenden Anwendungen muss eine Abteilung und bei standortübergreifenden muss ein Standort die Führung übernehmen“ (siehe A.6). Der Bedarf, sich regelmäßig auszutauschen und dies zu fördern wird also bei abteilungsübergreifenden und standortübergreifenden Anwendungen deutlich höher, wodurch man bei diesen Ausprägungen von einer starken Wirkbeziehung ausgehen kann.

In Bezug auf die unterschiedlichen Programmiererfahrungen lässt sich festzuhalten, dass „der gegenseitige Austausch auch wertvoll für die ist, die Erfahrung haben, um sich so die neuesten Erkenntnisse gegenseitig präsentieren zu können, Anwendungen zur Inspiration zu erhalten und sich gegenseitig bei Fragestellungen weiterzuhelfen“ (siehe A.8). Die Expert*innen empfehlen „viel zu kommunizieren und, dass die erfahrenen Entwickelnden bei den weniger erfahrenen Entwickelnden auf die Finger schauen“ (siehe A.11), um sich gegenseitig zu unterstützen. Man kann also eine normale Wirkbeziehung zwischen Programmierfähigkeiten und Anspruchsgruppen feststellen. „Allerdings würde man den Austausch zwischen nur erfahrenen Entwickelnden anders angehen, als wenn auch weniger erfahrene Leute mit hineinkommen, weil die eher nochmal eine andere Perspektive haben“ (siehe A.5). Durch Low-Code soll insbesondere eine Kultur angestrebt werden, in der proaktive Citizen Developer aktiv sind (s. Binzer u. Winkler 2022). Außerdem lässt sich in den regelmäßigen Community-Treffen beobachten, „dass Leute ohne Erfahrung und mit wenig Erfahrung zusammenkommen und die IT-Spezialist*innen weniger vertreten sind“ (siehe A.8). Obwohl also ein Austausch über alle Programmierfähigkeiten hinweg angestrebt werden soll, ist bei keinen bis wenig vorhandenen Programmierfähigkeiten nochmal explizierter darauf zu achten, dass man aus den unterschiedlichen Erfahrungen lernt und sich weiterentwickeln kann. Folglich kann man zwischen Anspruchsgruppen und keiner sowie wenig vorhandenen Programmierfähigkeiten von einer starken Wirkbeziehung ausgehen.

7.3 Typenspezifische Wirkbeziehungen: Planen

In diesem Teil wird die Beziehung zwischen den Anwendungsfalltypen und den Regeln innerhalb der Dimension „Planen" beleuchtet. Dabei wird dargestellt, welche Maßnahmen sich aus den einzelnen Regeln ableiten lassen – entweder als typenunabhängig für alle Anwendungstypen (mit normaler Intensität) oder als typspezifisch zugeschnitten auf bestimmte Anwendungstypen (mit starker Intensität). Eine grafische Darstellung dieser zu analysierenden Wirkbeziehungen innerhalb der Dimension „Planen" ist in Abbildung 7-3 zu finden.

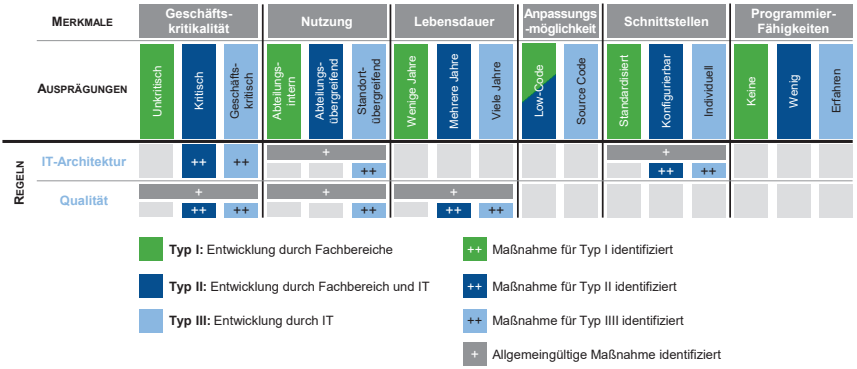

Abbildung 7-3: Wirkzusammenhänge in der Dimension „Planen" (eigene Darstellung)

IT-Architektur

Die Regel zur Verwaltung der IT-Architektur besagt: „Schnittstellen für die Integration in die bestehende IT-Architektur definieren und managen". Mithilfe der Regel soll sichergestellt werden, dass die Low-Code-Plattform bestmöglich in die bestehenden Strukturen integriert wird.

Neben den Schnittstellen haben auch die Geschäftskritikalität und die Nutzung Einfluss auf die Integration in die IT-Architektur. „Eine schlechte IT-Architektur kann im Zweifel dazu führen, dass die Definition der Single Source of Truth komplett kaputt geht. Dann ist das Vertrauen in die Daten nicht mehr hoch und der Nutzen der Anwendung geht ein bisschen verloren. Bei geschäftsunkritischen Fällen kann man vielleicht auch ein bisschen drüber hinwegsehen" (siehe A.5). Demnach ist es besonders relevant, für kritische und geschäftskritische Anwendungen, genau zu definieren, welche Anwendungen und Datenquellen angebunden werden und wie diese Daten verwaltet werden. Die Geschäftskritikalität in ihren Ausprägungen kritisch sowie geschäftskritisch haben folglich eine starke Wirkbeziehung zur IT-Architektur. Auch „bei Standortübergreifender IT-Architektur muss nochmal mehr abgestimmt werden" (siehe A.8), weil sich „die Standorte in Bezug auf ihre IT-Infrastruktur unterscheiden" (siehe A.6). Demnach hat die standortübergreifende Nutzung eine starke Wirkbeziehung zu der IT-Architektur.

In Bezug auf die Schnittstellen ist die Integration von Low-Code-Plattformen in bestehende IT-Systeme aufgrund von Schnittstellenlücken begrenzt (s. Binzer u. Winkler 2022). Die Expert*innen betonen, dass „eine Low-Code-Plattform zur Automatisierung von Geschäftsprozessen davon lebt, dass sie optimal eingebunden ist" (siehe A.6). Sie empfehlen, dass „grundsätzlich für die IT-Architektur immer Guidelines gemacht werden sollten" (siehe A.5) und dass man „zu Beginn einmalig einen Standard für Schnittstellen festlegen sollte" (siehe A.8). Demnach handelt es sich um einen normalen Wirkzusammenhang zwischen IT-Architektur und Schnittstellen. Bei konfigurierbaren Schnittstellen kann es bei der Schnittstellenerstellung zu Konfigurationsproblemen

kommen, die den Integrationsablauf zusätzlich behindern (s. ROKIS U. KIRIKOVA 2022). Auch gibt es Schnittstellen, die „im Standard nicht auf bestimmte Systeme zugreifen können“ (siehe A.8), wodurch es sich hier lohnen würde, konfigurierbare Schnittstellen aufzusetzen. Dies zu prüfen, mündet in einer weiteren Maßnahme, wodurch man bei konfigurierbaren Schnittstellen und der IT-Architektur von einer starken Wirkbeziehung ausgehen kann. Bei individuellen Schnittstellen „sollte die IT-Architektur dann noch mal mehr berücksichtigt werden“ (siehe A.5). „Wenn man es komplett individualisiert, hat die Schnittstelle einen konfigurierbaren Teil, der die individuelle Funktion steuert“ (siehe A.5). Folglich kann man auch hier von einer starken Wirkbeziehung zwischen IT-Architektur und individuellen Schnittstellen ausgehen.

Qualität

Die Regel zur Qualitätssicherung sieht vor, dass ein Qualitätssicherungsprozess mit Testing Framework für die entwickelten Anwendungen eingeführt werden soll. Die Expert*innen raten dazu, die „Qualität niemals ganz zu vernachlässigen“ (siehe A.5), und betonen, dass „je höher die Auswirkungen von Fehlern sein können, desto wichtiger ist es auch, auf Qualität zu achten“ (siehe A.5). Demnach ist es auch für unkritische, abteilungsinterne Anwendungen mit kurzer Lebensdauer wichtig, Qualitätskontrollen durchzuführen, wodurch man bei Geschäftskritikalität, Nutzung und Lebensdauer von einem einfachen Wirkzusammenhang ausgehen kann. Je nach Art der Anwendung kann es darüber hinaus empfehlenswert sein, dass IT-Expert*innen eine abschließende Kontrolle durchführen (s. HEUER ET AL. 2022). „Je kritischer ein Prozess wird, desto mehr guckt dann auch die IT drauf“ (siehe A.8) und „desto mehr zusätzliche Qualitätsmaßnahmen werden hinzugefügt“ (siehe A.9). Bezogen auf die Geschäftskritikalität „muss das Testen bei kritischen und geschäftskritischen Prozessen mehr beachtet werden, damit auch alles glatt läuft“ (siehe A.6). Besonders „für geschäftskritische Anwendungen werden mehr Tests durchgeführt, auch automatisiert“ (siehe A.9). Demnach sind bei diesen Anwendungen mehr Maßnahmen der Qualitätssicherung notwendig, wodurch man bei kritischen und geschäftskritischen Anwendungen von einer starken Wirkbeziehung zur Qualität ausgehen kann. „Gleiches gilt für die Nutzung“ (siehe A.5). Bei abteilungsinternen Entwicklungen „sind die Entwickelnden höchstwahrscheinlich die Nutzer*innen oder stehen den Nutzer*innen sehr nahe“ (siehe A.9), wodurch sich die Fehler schnell beheben lassen. „Je größer das Team ist, je höher ist die Auswirkung des Fehlers“ (siehe A.9). Daher empfiehlt es sich „bei abteilungsübergreifenden und standortübergreifenden Anwendungen besonders dann noch mal, eigene interne Qualitätsprozesse drüber laufen zu lassen“ (siehe A.5). Folglich kann man bei abteilungsübergreifenden und standortübergreifenden Anwendungen von einer starken Wirkbeziehung ausgehen. Das gleiche Verhalten der Wirkbeziehungen lässt sich bei der Lebensdauer festhalten. „Wenn die Anwendung kurzfristig gebaut wird, macht man sich weniger Sorgen um die Qualität“ (siehe A.9). Bei einer kurzen Lebensdauer „müssen der Aufwand und der Nutzen im Verhältnis stehen“ (siehe A.5), wodurch weitere Qualitätsmaßnahmen insbesondere bei mittel- bis langfristigen Anwendungen aufgebaut werden müssen und dort eine starke Wirkbeziehung besteht.

7.4 Typenspezifische Wirkbeziehungen: Aufbauen

Dieser Abschnitt widmet sich eingehend der Untersuchung der Beziehungen zwischen verschiedenen Anwendungsfalltypen und den zugehörigen Regeln in der „Aufbauen"-Dimension. Es wird verdeutlicht, welche spezifischen Maßnahmen sich aus den Regeln ableiten – generell gültig für alle Typen von Anwendungsfällen (mit normaler Intensität) oder maßgeschneidert für ausgewählte Typen von Anwendungsfällen (mit erhöhter Intensität). Eine übersichtliche grafische Darstellung der zu analysierenden Zusammenhänge in der „Aufbauen"-Dimension ist in Abbildung 7-4 ersichtlich.

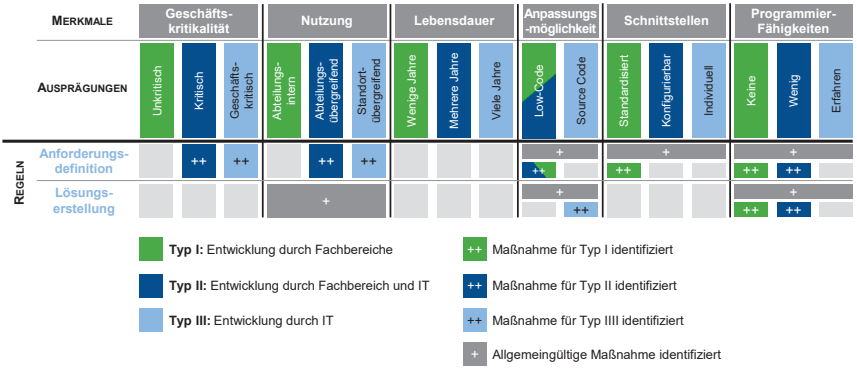

Abbildung 7-4: Wirkzusammenhänge in der Dimension „Aufbauen" (eigene Darstellung)

Anforderungsdefinition

Die Regel zur Anforderungsdefinition gibt vor, dass ein Anforderungsmanagementprozess und Kriterien für die Anforderungsdefinition von Low-Code-Anwendung aufgesetzt werden sollten, um sicherzustellen, dass die Anforderungen sauber definiert werden und diese auch durch die Low-Code-Plattform umgesetzt werden können. „Je kritischer die Anwendung ist, desto wichtiger ist es, dass Anforderungen sauber definiert sind" (siehe A.5). Daher sollte „bei kritischen und geschäftskritischen Anwendungen nochmal besser geprüft werden, ob das wirklich die richtigen Anforderungen sind und ob man sie auch wirklich umsetzen kann" (siehe A.8). Folglich kann man bei kritischen und geschäftskritischen Anwendungen von einem starken Zusammenhang zur Anforderungsdefinition ausgehen. Auch in Bezug auf die Nutzergruppe ist es wichtig, „frühestmöglich alle Beteiligten an einen Tisch zu bekommen, um lange Abstimmungszyklen zu vermeiden" (siehe A.7). Die Abstimmung zu den Anforderungsdefinitionen ist abteilungsintern einfacher, weil „man jeden Tag miteinander zusammenarbeitet und eine gemeinsame Sprache hat. Spätestens ab abteilungsübergreifend, wird es dann intensiver" (siehe A.6). Expert*innen empfehlen ein geregelteres Vorgehen bei abteilungsübergreifenden und standortübergreifenden Anwendungen. „Abteilungsinterne Anwendungen können die Abteilungen gut organisieren, das muss auch nicht ein hochformalisierter Prozess sein. Aber abteilungsübergreifend und standardübergreifend definitiv" (siehe A.5). Dazu kann ein Tool genutzt werden, „wo der Fachbereich

die Möglichkeit hat, Ideen mit einzuspeisen“ (siehe A.6). Demnach kann man bei abteilungsübergreifenden und standortübergreifenden Anwendungen von einem starken Wirkzusammenhang zur Anforderungsdefinition sprechen.

In Bezug auf die Anpassungsmöglichkeiten sollte zu Beginn geprüft werden, „ob sich der Anwendungsfall dazu eignet, mit Low-Code umgesetzt zu werden“ (siehe A.7). Abhängig von der Komplexität der Anforderungen und den geforderten Leistungsmerkmalen kann es zudem vorkommen, dass die technischen Möglichkeiten von Low-Code nicht ausreichen, um alle Anforderungen zu erfüllen (s. Luo et al. 2021). Demnach besteht ein einfacher Wirkzusammenhang zwischen Anpassungsfähigkeit und Anforderungsdefinition. Darüber hinaus empfehlen die Expert*innen, bei der Anforderungsdefinition zu prüfen, ob „die Anforderungen mit bestehenden Bausteinen erfüllt werden“ (siehe A.5) können. Wenn nur Low-Code-Bausteine zur Verfügung stehen, ist diese Maßnahme besonders wichtig, wodurch man von einer starken Wirkbeziehung zwischen Anforderungsdefinition und der Anpassung nur durch Low-Code ausgehen kann. Ein einfacher Wirkzusammenhang besteht zwischen Anforderungsdefinition und Schnittstellen. Bei der Anforderungsdefinition sollte geprüft werden, „ob alle notwendigen Schnittstellen entlang des Gesamtprozesses vorhanden sind oder ob der Bedarf besteht, konfigurierbare oder wirklich individuelle Schnittstellen zu implementieren“ (siehe A.5). Ein starker Zusammenhang ergibt sich dann noch aus der Anforderungsdefinition und den standardisierten Schnittstellen, weil geprüft wird, ob die definierten Anforderungen „mit standardisierten Schnittstellen umgesetzt werden können“ (siehe A.8).

In Bezug auf die Programmierfähigkeiten empfehlen die Expert*innen, bei der Anforderungsdefinition zu überprüfen, welche Programmierfähigkeiten benötigt werden. Unerfahrene Entwickelnde „bauen sich die Anwendung in einem Low-Code-Tool mit weniger Funktionalitätsumfang“ (siehe A.6). Demnach kann nicht jede Anwendung mit unerfahrenen Entwickelnden gebaut werden. Man sollte vor der Lösungsentwicklung prüfen, „ob sich der Anwendungsfall für Citizen Developer eignet“ (siehe A.7). Demnach kann man hier einen einfachen Zusammenhang zwischen Programmierfähigkeiten und Anforderungsdefinition festhalten. Ferner besteht das Risiko, dass die Anforderungen nicht ausreichend klar formuliert sind (s. Rokis u. Kirikova 2022). „Bei der Anforderungsdefinition ist es wichtig zu verstehen, was eine Anforderung, für Auswirkungen mit sich bringt“ (siehe A.7). „Mit mehr Erfahrungen kann man die Anforderungen auch besser einschätzen“ (siehe A.8). Dadurch sind mehr Maßnahmen bei der Anforderungsüberprüfung notwendig, wenn diese mit keiner oder wenig Programmiererfahrung geschrieben wurden. Demnach ist hier ein starker Zusammenhang vorhanden.

Lösungserstellung

Die Regel zur Lösungserstellung stellt sicher, dass bei der Anwendungsentwicklung Standards für das User Interface (UI) und die entwickelten Workflows genutzt werden und sorgt so für einheitliche Anwendungen und die Nutzung von wiederverwertbaren Komponenten. Es ist wichtig, einen Standard bei den Anwendungen zu setzen, „sonst

werden die Mitarbeiter*innen irgendwann nicht mehr mitspielen“ (siehe A.6). Bei der Entwicklung von Standards ist es wichtig, dass die gesamte Nutzergruppe mit einbezogen wird. Auch die Expert*innen wissen aus Erfahrung, dass „bei der Entwicklung des Standards für unseren Standort mehrere Abteilungen miteinbezogen wurden“ (siehe A.8). Demnach besteht ein normaler Wirkzusammenhang zwischen der Lösungserstellung und der Nutzung.

In Bezug auf die Anpassungsfähigkeit der Low-Code-Plattform empfiehlt es sich, „ein Designsystem, das eine breite Palette von Standard-UI-Komponenten enthält“ (siehe A.9), zu entwickeln. Das Designsystem kann „von den lokalen Entwicklern und auch von den Produkteigentümern wiederverwendet werden“ (siehe A.9). Das Erstellen von Standardkomponenten ist ein „Prozess von Neuerungen, der mitgedacht werden muss“ (siehe A.5). Folglich kann man von einer normalen Wirkbeziehung zwischen Lösungserstellung und Anpassungsfähigkeiten ausgehen. Wenn die Low-Code-Plattform auch nutzerseitig im Source Code angepasst werden kann, „ist es ein Riesenaufwand, wenn man ein einheitliches Design pflegen muss, dann muss man das ja auch alles definieren an irgendwelchen Stellen“ (siehe A.7). Daher „braucht es so eine Art Regelkatalog“ (siehe A.5), um zu entscheiden, wann etwas nutzerseitig im Source Code ausgebaut wird. Demnach kann man bei Source Code Anpassungen von einer starken Wirkbeziehung zu der Lösungserstellung ausgehen.

Bei den Programmierfähigkeiten lässt sich sowohl für erfahrene Entwickelnde als auch für unerfahrene Entwickelnde feststellen, dass teilweise keine Standards verwendet werden. „Erfahrene Programmierer programmieren vieles nativ und nutzen kaum die vorgefertigten Module“ (siehe A.7), während es bei unerfahrenen Entwickelnde „ein kritisches Ding ist, dass die Komplexität der Lösung zum Problem passt“ (siehe A.11), die Lösung also teilweise zu komplex gebaut wird. Demnach empfiehlt es sich, für alle Level der Programmierfähigkeiten „Modellierungsrichtlinien zu definieren, die besagen, wie man seine Anwendung erstellt und wie man Namenskonventionen anwendet“ (siehe A.9). Darüber hinaus sollte man die entwickelten Anwendungen von erfahrenen Low-Code-Entwicklern überprüfen lassen. Es handelt sich demnach um eine normale Wirkbeziehung.

7.5 Typenspezifische Wirkbeziehungen: Ausführen

In diesem Teil wird die Beziehung zwischen den Anwendungsfalltypen und den Regeln innerhalb der Dimension „Ausführen" beleuchtet. Dabei wird dargestellt, welche Maßnahmen sich aus den einzelnen Regeln ableiten lassen – entweder als typenunabhängig für alle Anwendungstypen (mit normaler Intensität) oder als typspezifisch zugeschnitten auf bestimmte Anwendungstypen (mit starker Intensität). Eine grafische Darstellung dieser zu analysierenden Wirkbeziehungen innerhalb der Dimension „Ausführen" ist in Abbildung 7-5 zu finden.

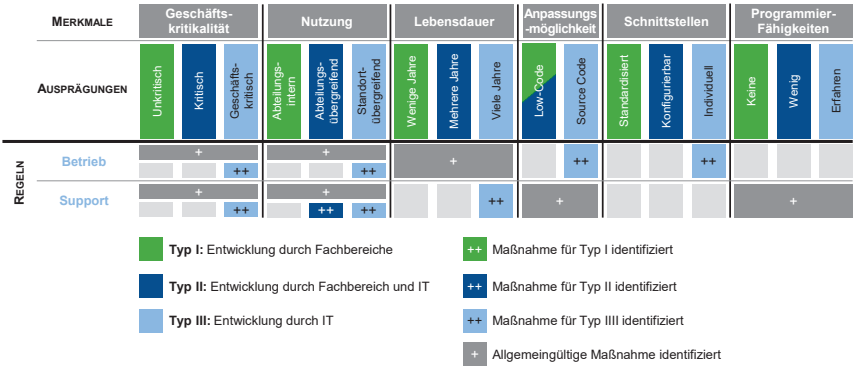

Abbildung 7-5: Wirkzusammenhänge in der Dimension „Ausführen" (eigene Darstellung)

Betrieb

Die Regel für den Betrieb von Low-Code-Anwendung sieht vor, dass ein Bereitstellungsprozess festgelegt wird und die Anwendungen durch Versionskontrollen kontinuierlich verbessert werden. Je nach Anwendung kann der „gesamte Deployment Prozess mitsamt Versionierung so weit wie möglich automatisiert" (siehe A.6) werden. Bei der Geschäftskritikalität der Anwendungen kann man festhalten, dass „je kritischer der Prozess ist, so detailliert muss der Bereitstellungsprozess definiert werden" (siehe A.8). Folglich lassen sich für die unterschiedlichen Kritikalitätsstufen unterschiedliche Maßnahmen abteilen. „Wenn es unkritisch ist, muss man nur darauf achten, dass es einen klaren Kommunikationsprozess gibt" (siehe A.5). Da es für jede Art der Anwendung relevant ist zu kommunizieren, was der Funktionalitätsumfang der Anwendung ist und ob diese Anwendung etwas ablöst, kann man von einem normalen Wirkzusammenhang zwischen Geschäftskritikalität und Betrieb ausgehen. „Bei wirklich geschäftskritischen Anwendungen muss man ganz klar definieren, wann etwas verändert wird und ob man andere Themen gleichzeitig betreiben muss" (siehe A.5). Demnach besteht eine starke Intensität zwischen geschäftskritischer Anwendung und deren Betrieb. „Das Gleiche kann man auch so bei der Nutzung übernehmen" (siehe A.5). Die Expert*innen empfehlen, dass die Verantwortung des Betriebs „am besten das Team trägt, das auch dieses Produkt baut" (siehe A.10). „Je mehr Abteilungen die Anwendung nutzen, umso komplexer wird der Prozess, die Anwendung einzuführen" (siehe A.8), wodurch man hier die gleichen Maßnahmen wie bei der Geschäftskritikalität ableiten kann (siehe A.5). Es besteht ein starker Zusammenhang zwischen standortübergreifender Nutzung und Betrieb. In Bezug auf die Lebensdauer empfehlen die Expert*innen, „wenn die Anwendung nur ein Jahr benutzt wird, dann kann auch weniger in die Wartung investiert werden" (siehe A.9). Je nach Lebensdauer kann man also abwägen, wie viel Zeit in den Betrieb investiert wird. Es handelt sich um eine normale Wirkbeziehung.

Darüber hinaus lässt sich festhalten, dass das Durchführen von Upgrades bei größerer Individualisierbarkeit komplexer wird. „Wenn Änderungen vorgenommen werden müssen, muss man sich mit dem speziell angefertigten Source Code auseinandersetzen“ (siehe A.9). Ein ähnliches Verhalten zeigt sich bei den individuellen Schnittstellen. „Je mehr individuelle Schnittstellen ich habe, desto komplexer wird der Betrieb“ (siehe A.8). Bei individuellen Schnittstellen "entsteht eine Art von Overhead“ (siehe A.9), wodurch man bei der Source Code Anpassbarkeit und den individuellen Schnittstellen von einer starken Wirkbeziehung ausgehen kann.

Support

Die Regel zu Support sieht vor, dass Verantwortlichkeiten für die Wartungsaufgaben je Anwendung festgelegt werden, um so einen störungsfreien Betrieb der Anwendungen sicherzustellen. In Bezug auf die Geschäftskritikalität lässt sich festhalten, dass man „auch unkritische Prozesse nicht einfach so liegen lassen darf“ (siehe A.5), wodurch man von einer normalen Wirkbeziehung zwischen Geschäftskritikalität sowie Support ausgehen kann. Ähnlich wie beim Betrieb lässt sich auch hier festhalten, dass „je kritischer der Prozess ist, umso detaillierter muss der Supportprozess definiert werden“ (siehe A.8). In diesem Prozess sollten „die geschäftskritischeren Anwendungen im Support höher priorisiert“ (siehe A.7) werden, weil „die Toleranz für Ausfallzeiten sinkt“ (siehe A.5). Für geschäftskritische Anwendungen empfehlen die Expert*innen, dass man „einen eigenen Support“ aufbaut (siehe A.9). Demnach besteht eine starke Wirkbeziehung von geschäftskritischen Anwendungen zu Support. Ähnliche Wirkzusammenhänge lassen sich bei der Nutzung feststellen. Bei abteilungsinternen Anwendungen ist es ausreichend, „den Support in der Abteilung, die die Anwendung erstellt“ (siehe A.9) zu belassen, während man für abteilungsübergreifende und standortübergreifende Anwendungen „einen eigenen Support“ (siehe A.9) aufbauen sollte. Demnach besteht ein normaler Zusammenhang von Nutzung zu Serviceanfragen und Störung, während ein starker Zusammenhang zu abteilungs- und standortübergreifenden Anwendungen besteht. In Bezug auf die Lebensdauer sollte man bei Anwendungen, die „sehr lange im Betrieb sind, noch mal intensiver in die Schulung des Servicepersonals gehen“ (siehe A.5). Demnach lässt sich hier eine starke Wirkbeziehung zwischen langer Lebensdauer und Support feststellen.

Auch „die Art der Plattformanpassung hat einen Einfluss auf den Support“ (siehe A.5). Je nachdem, welcher Fehler aufkommt, liegt das Beheben des Fehlers „in der internen oder externen Verantwortung“ (siehe A.5). Es besteht also ein normaler Zusammenhang zwischen Anpassungsfähigkeit und Support. Auch bei den Programmierfähigkeiten lässt sich „mit mehr Erfahrungen ein besserer Support leisten“ (siehe A.8). Es gilt zu untersuchen, ob man „Expert*innen braucht, um das Problem zu lösen“ (siehe A.5), oder ob man andere Möglichkeiten nutzt. Mehrere Möglichkeiten des Supports existieren, zuerst beginnt man mit einem Forum und der Dokumentation, dann kommen die Support Sessions. In denen zuerst versucht wird, das Problem mit erfahrenen internen Entwickelnden zu lösen“ (siehe A.7), bevor eine externe Support Session gebucht wird. Es empfiehlt sich also, mehrere Möglichkeiten des Supports aufzubauen und die Supportanfragen je nach Problem und Entwicklungserfahrung unterschiedlich zu

routen. Demnach besteht ein normaler Wirkzusammenhang zwischen Programmiererfahrung und Support.

7.6 Typenspezifische Wirkbeziehungen: Überwachen

In diesem Teil wird die Beziehung zwischen den Anwendungsfalltypen und den Regeln innerhalb der Dimension „Überwachen" beleuchtet. Dabei wird dargestellt, welche Maßnahmen sich aus den einzelnen Regeln ableiten lassen – entweder als typenunabhängig für alle Anwendungstypen (mit normaler Intensität) oder als typspezifisch zugeschnitten auf bestimmte Anwendungstypen (mit starker Intensität). Eine grafische Darstellung dieser zu analysierenden Wirkbeziehungen innerhalb der Dimension „Überwachen" ist in Abbildung 7-6 zu finden.

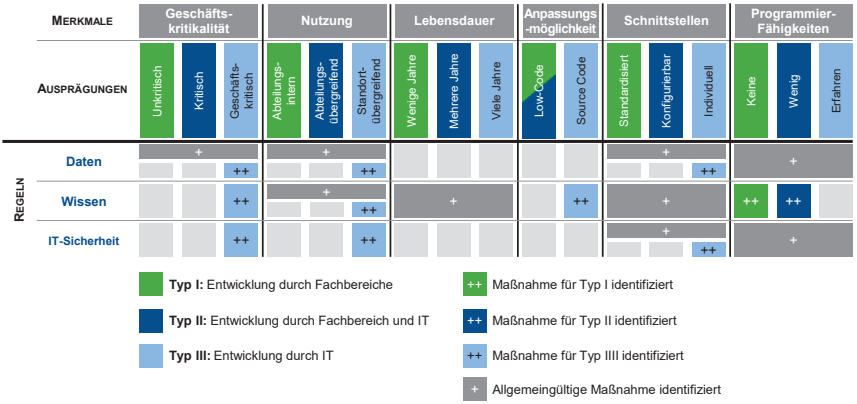

Abbildung 7-6: Wirkzusammenhänge in der Dimension „Überwachen" (eigene Darstellung)

Daten

Die Regel zu Daten hat zum Ziel, dass der Umgang mit Daten durch ein Zugriffsmanagement eingeschränkt wird, weil die Entwicklung von Software mithilfe von Low-Code-Plattformen Entwickelnden einen unkomplizierten Zugriff auf verschiedene Arten von Daten ermöglicht (s. PRINZ ET AL. 2022; ROKIS U. KIRIKOVA 2022). „Das Thema Geschäftskritikalität ist häufig mit den integrierten Daten verbunden, weil sich die Daten auf die Anwendungen beziehen“ (siehe A.7). Die Expert*innen empfehlen „auch bei unkritischen Anwendungen ein Regelwerk für Daten“ einzuführen (siehe A.5). Folglich kann man von einer normalen Wirkbeziehung zwischen Daten und Geschäftskritikalität ausgehen. Einen besonderen Fokus sollte man allerdings noch auf die geschäftskritischen Anwendungen legen. Denn „bei den geschäftskritischen Anwendungen geht vielmehr um die Sensibilität von Informationen“ (siehe A.6). Die Expert*innen empfehlen hier nochmal, „besonders darauf zu achten, wer welche Daten sieht und diese verändern darf“ (siehe A.5). Demnach besteht zwischen geschäftskritischen Anwendungen und Daten eine starke Wirkbeziehung. Ein ähnliches Verhalten lässt sich bei der

Nutzung beobachten. Bei jeder Anwendung sollte man „definieren, welche Daten wem zur Verfügung stehen“ (siehe A.5). Das ist „je nach Unternehmensstruktur“ (siehe A.6) anders. Demnach kann man von einem normalen Zusammenhang zwischen Daten und Nutzung ausgeben. Zusätzliche Maßnahmen fallen bei der standortübergreifenden Nutzung an. „Wenn Standort übergreifend etwas genutzt wird, muss das Zugriffsmanagement nochmal detaillierter verwaltet werden“ (siehe A.8). Es gilt zu überprüfen, ob „beispielsweise Standort A die Lagerbestände vom Standort B sehen darf“ (siehe A.6). Demnach besteht hier ein starker Zusammenhang.

Die Verwaltung von Daten kann außerdem durch die Bereitstellung von konformen Schnittstellen erfolgen (s. HEUER ET AL. 2022), weil „Schnittstellen einen Zusammenhang zu Daten haben“ (siehe A.5) und man den Zugriff auf Schnittstellen gegebenenfalls einschränken kann. Die Expert*innen empfehlen, sich „Gedanken darüber zu machen, auf welche Daten mit welchen Schnittstellen zugegriffen werden kann und ob es sich dabei um sensible Informationen handelt“ (siehe A.9). In der Regel gibt es „unterschiedliche Personen, die die Schnittstellen administrativ betreuen“ (siehe A.7) und das Zugriffsmanagement verwalten. Standardisierte und konfigurierbare Schnittstellen „können ohne Probleme genutzt werden“ (siehe A.8). Sobald es sich um individuelle Schnittstellen handelt, „muss man Datenvalidierungsregeln und dergleichen erstellen“ (siehe A.9). Expert*innen sagen, die individuellen Schnittstellen „müssen beantragt werden“ (siehe A.6). Durch diese weitere Maßnahme kann man hier von einer starken Intensität sprechen. In Bezug auf die Programmierfähigkeiten lässt sich festhalten, dass „weniger erfahrene Softwareentwickler*innen nicht die Fähigkeiten haben, alle Daten direkt verwenden zu können“ (siehe A.9). Bei Daten aus manchen Systemen muss man sich beispielsweise „mit SQL-Skripten aushelfen“ (siehe A.6). Es empfiehlt sich also, bei den Programmierfähigkeiten zu überprüfen, wer in der Lage ist, auf welche Daten zuzugreifen. Demnach lässt sich hier von einer normalen Intensität der Wirkbeziehen sprechen.

Wissen

Die Regel zum Wissensmanagement zielt darauf ab, eine Übersicht und Dokumentation der Funktionsweise von allen erstellten Anwendungen zu erstellen. Eine umfassende Dokumentation der Anwendungen ist von entscheidender Bedeutung, um ihre langfristige Nutzbarkeit und Erweiterbarkeit sicherzustellen (s. INDAMUTSA ET AL. 2021). Bei geschäftskritischen Anwendungen „ist natürlich auch immer das Interesse da, dass Software dann vernünftig reagiert“ (siehe A.11) und gewartet werden kann. „Je kritischer ein Prozess wird, desto mehr muss ja dokumentiert werden“ (siehe A.8). Denn die Dokumentation kann dabei helfen „diese Veränderung zu tracken“ (siehe A.10) und die Anwendung schneller anzupassen. Demnach empfiehlt es sich, die Funktionen im Code besonders sauber zu dokumentieren. Man kann also von einer starken Wirkbeziehung ausgehen. In Bezug auf die Nutzung empfehlen die Expert*innen Verantwortlichkeiten für Themen festzulegen. Als Beispiel wird eine Matrix genannt, die festlegt, dass „Person XY verantwortlich für Thema XY ist“ (siehe A.7). Es handelt sich also um eine normale Wirkbeziehung. Zwischen standortübergreifenden Anwendungen und dem Wissensmanagement wird empfohlen, „nochmal eine Sonderregel einzuführen“

(siehe A.5) und die Verantwortlichkeiten besonders zu überprüfen und festzulegen. Demnach ist hier ein starker Wirkzusammenhang vorhanden. Bei der Lebensdauer lässt sich festhalten, dass es unabhängig von der Lebensdauer „immer eine gute Idee“ (siehe A.10) ist, zu dokumentieren. Es kann vorkommen, dass beispielsweise „Studenten Dinge entwickeln und da hat man ein Problem, wenn die weg sind“ (siehe A.8). Je nach Lebensdauer kann man aber abwägen, „was dokumentiert werden muss und was sich nicht lohnt“ (siehe A.10). Darüber hinaus muss dafür gesorgt werden, „dass sich frühzeitig um eine Übergabe gekümmert wird“ (siehe A.8). Demnach kann man hier von einem normalen Zusammenhang zwischen Lebensdauer und Wissen sprechen.

Bei der Anpassungsfähigkeit liegt der Fokus auf der nutzerseitigen Erweiterung des Source Codes. „Sobald man eine Erweiterung implementiert hat, sollte diese als eine neue Low-Code-Komponente erstellt werden“ (siehe A.9). Hier sollte besonders darauf geachtet werden, dass „die Lösungsbausteine so dokumentiert sind, dass sie für alle Anwendenden nachvollziehbar sind“ (siehe A.5). Man kann also von einer starken Wirkbeziehung zwischen Source Code-Anpassungen und Wissen ausgehen. Auch bei den Schnittstellen „braucht man eine formale Definition, die angibt, was rein und was raus geht“ (siehe A.11). Je nach Schnittstelle kann diese Dokumentation dann anders gestaltet sein. „Bei einer standardisierten Schnittstelle muss zwar dokumentiert werden, aber je individueller es wird, desto höher muss da auch der Dokumentationsaufwand sein.“ Da die Dokumentation aber für alle Schnittstellen relevant ist, kann man hier von einem normalen Zusammenhang zwischen Wissen und Schnittstellen ausgehen. In Bezug auf die Programmierfähigkeiten sollte die Dokumentation so gestaltet sein, dass „jemand mit wenig oder ohne Entwicklungskenntnisse etwas mit der Dokumentation anfangen kann“ (siehe A.5). Es besteht also eine starke Wirkbeziehung zwischen Wissen und keiner bzw. wenig Programmierfähigkeiten.

IT-Sicherheit

Die Regel zu IT-Sicherheit hat das Ziel, sowohl organisatorische als auch technische Sicherheitspraktiken umzusetzen, um die Anwendungen sicher betreiben zu können. Bei geschäftskritischen Anwendungen empfehlen die Expert*innen zusätzliche Sicherheitsmaßnahmen. „Zum Beispiel werden geschäftskritische Anwendungen von weiteren dritten Sicherheitsfirmen Pen getestet“ (siehe A.10). Demnach besteht ein starker Wirkzusammenhang zwischen geschäftskritischen Anwendungen und IT-Sicherheit. Bei der Nutzung lässt sich festhalten, dass „wenn das nur Software ist, die intern läuft, man das durchaus kleiner und einfacher handhaben kann“ (siehe A.10). Bei standortübergreifenden Anwendungen empfehlen die Expert*innen weitere Sicherheitsmaßnahmen. „Die Standorte haben vielleicht unterschiedliche Richtlinien in Richtung Sicherheitspraktiken. Das ist zumindest im EU-Ausland und in der EU unterschiedlich und in Deutschland und im deutschen Ausland auch“ (siehe A.8). Demnach besteht ein starker Zusammenhang zwischen standortübergreifenden Anwendungen und der IT-Sicherheit.

Ein weiterer großer Aspekt der IT-Sicherheit ist, „dass die Daten nicht manipulierbar und nicht einsehbar übertragen werden“ (siehe A.11). Für die Schnittstellen empfehlen

Expert*innen, aus Sicherheitsgründen die Schnittstellen „zu verschlüsseln, ganz egal, ob die Schnittstelle jetzt standardisiert, konfigurierbar, oder individuell ist“ (siehe A.5). Sobald man sich dazu entscheidet, eine API oder eine andere Schnittstelle nach außen anzubieten“ (siehe A.10), sollte sichergestellt werden, „dass die API gekappt werden kann, zum Beispiel durch API-Gateways oder ähnliche Mechanismen“ (siehe A.10). In diesem Zusammenhang empfiehlt es sich auch, externe Schnittstellen zu überwachen (s. HEUER ET AL. 2022). Dies kann gewährleistet werden, indem man überlegt, ob „es irgendwelche Alarme gibt, die man nutzen kann, um direkt zu erkennen, ob da ein Problem ist“ (siehe A.5). Ein besonderer Fokus liegt auf individuellen Schnittstellen. „Die Erstellung der individuellen Schnittstellen hängt von den Systemen und deren Sicherheitsanforderungen ab“ (siehe A.7). Es empfiehlt sich daher, „bei individuellen Schnittstellen doppelt zu kontrollieren, weil man etwas baut, was noch nicht vorhanden ist“ (siehe A.5). Folglich existieren viele Maßnahmen, die für alle Arten von Schnittstellen notwendig sind, sowie weitere Maßnahmen explizit für individuelle Schnittstellen. Man kann also von einem normalen Wirkzusammenhang zwischen Schnittstellen und IT-Sicherheit ausgehen, sowie von einer starken Wirkbeziehung zu individuellen Schnittstellen. In Bezug auf die Programmierfähigkeiten lässt sich festhalten, dass für die Implementierung von Sicherheitspraktiken besondere Kenntnisse vorhanden sein müssen. Daher empfehlen die Expert*innen, „die Fachbereiche nie selbst für die Anwendungen verantwortlich zu machen“ (siehe A.6). Sollten die Kenntnisse in der Organisation nicht vorhanden sein, kann man „das, was etabliert werden muss, halbwegs gut auslagern“ (siehe A.10). Es gilt also immer zu überprüfen, was intern geleistet werden kann und wo man externe Unterstützung benötigt. Demnach kann man von einem normalen Zusammenhang zwischen IT-Sicherheit und Programmierfähigkeiten ausgehen.

7.7 Reflexion und Zusammenfassung der Ergebnisse

In den vorherigen Unterkapiteln wurde ein methodischer Ansatz zur Untersuchung von Wirkbeziehungen erläutert. Durch die Verknüpfung der ersten beiden Beschreibungsmodelle entstand eine Matrix, die die Wechselwirkungen zwischen den in Kapitel 5 hergeleiteten Low-Code-Anwendungsfällen und den in Kapitel 6 hergeleiteten Regeln für Low-Code verdeutlicht. Eine Wirkbeziehung zwischen Low-Code-Regel und Anwendungsfalltyp ergab sich genau dann, wenn sich aus dem Merkmal oder der Ausprägung des Anwendungsfalltyps eine Maßnahme zum Einhalten der entsprechenden Low-Code-Regel ableiten ließ.

Für die Einschätzung der Wirkintensitäten wurden gezielt Expert*innen aus unterschiedlichen Fachbereichen und Industriezweigen sowie mit Erfahrungen in verschiedenen Low-Code-Anwendungsfalltypen konsultiert. Trotz der Vielfalt der Expert*innenmeinungen zeigte sich ein weitgehend einheitliches Bild der Wirkzusammenhänge. Die Intensität der Wechselwirkungen gibt Aufschluss darüber, ob die Maßnahme zum Einhalten der Regel für alle Anwendungsfalltypen gültig ist, oder ob zur Regeleinhaltung spezifische Maßnahmen für einen bestimmten Anwendungsfalltyp notwendig sind. Lässt sich eine Maßnahme zum Einhalten der Regel aus dem Merkmal des

Anwendungsfalltyps ableiten, handelt es sich um eine normale Intensität der Wirkbeziehung und die Maßnahme ist typenunabhängig. Lässt sich eine Maßnahme zum Einhalten der Regel aus der Merkmalsausprägung ableiten, handelt es sich um eine starke Intensität der Wirkbeziehung und die Maßnahme ist typspezifisch. Diese Analyse bot erstmalig einen umfassenden Überblick über das Zusammenspiel verschiedener Low-Code-Anwendungsfälle und Regeln. Abbildung 7-7 zeigt die Ergebnisse der Untersuchung.

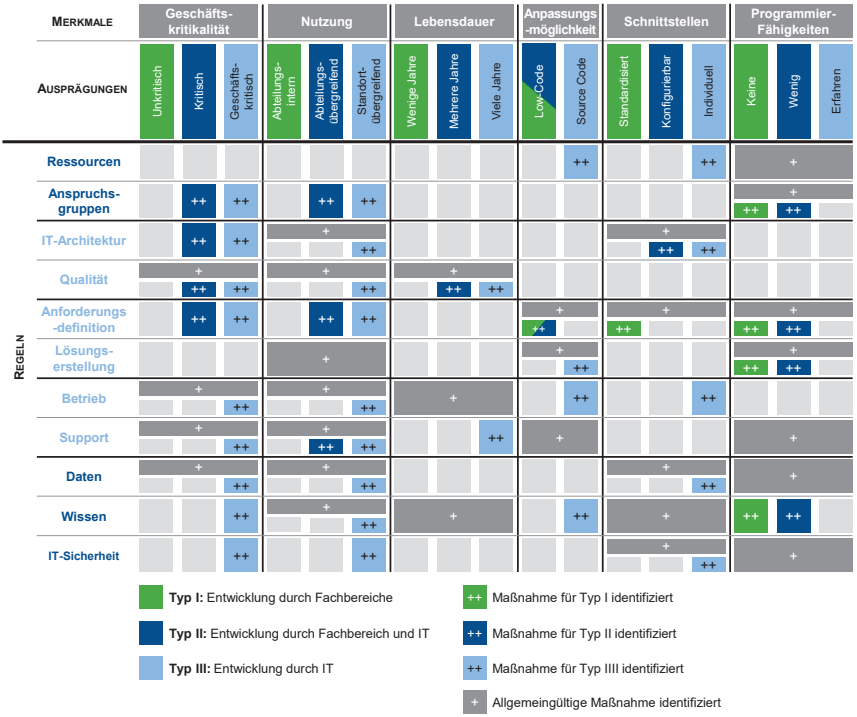

Abbildung 7-7: Wirkzusammenhänge zwischen Anwendungsfalltypen und Low-Code-Regeln (eigene Darstellung)

Es wurde deutlich, dass einige Maßnahmen zur Einhaltung der Low-Code-Regeln unabhängig vom Anwendungstyp allgemeingültig sind, während andere Maßnahmen speziell für einen oder mehrere bestimmte Low-Code-Anwendungsfalltypen notwendig sind. Die Ergebnisse adressieren die in Kapitel 3.3 identifizierte Forschungslücke und beantworten die dritte Forschungsfrage:

Wie können aus den Low-Code-Anwendungsfällen Wirkbeziehungen zu den Low-Code-Regeln erklärt werden?

Nachfolgend wird ein Vorgehen beschrieben, wie das Low-Code-Regelwerk durch allgemein übertragbare Gestaltungsmaßnahmen, abhängig von den verschiedenen Low-

Code-Anwendungstypen, eingehalten werden kann. Die Gestaltungsmaßnahmen liefern Handlungsempfehlungen, wie die bestehende IT-Governance um das Low-Code-Regelwerk sinnvoll ergänzt werden kann.

8 Gestaltung des Regelwerks

Das nachfolgende Kapitel dient der Beantwortung der vierten untergeordneten Forschungsfrage:

Wie kann das Vorgehen zur anwendungsfallbasierten Definition eines Regelwerks für Low-Code in produzierenden Unternehmen gestaltet werden?

Es wird untersucht, mit welchen Gestaltungsmaßnahmen das in Kapitel 6 beschriebene Regelwerk anwendungsfallbasiert eingehalten werden kann. Nachdem in Kapitel 7 die Wirkbeziehungen zwischen den Low-Code-Anwendungstypen und den Low-Code-Regeln identifiziert wurden, sollen nun geeignete Gestaltungsmaßnahmen zur Einhaltung dieser Regeln für jeden Anwendungstyp festgelegt werden. Die in Kapitel 7 durch Expert*inneninterviews identifizierten Maßnahmen basieren auf den praktischen Erfahrungen der befragten Expert*innen. Daher werden in Kapitel 8.1 aus diesen spezifischen Maßnahmen übertragbare Gestaltungsempfehlungen abgeleitet und detailliert erläutert. Kapitel 8.2 widmet sich der praktischen Anwendung des Regelwerks, indem ein Vorgehen für die Anwendung des Regelwerks vorgestellt wird, was das Ziel hat, die bestehende IT-Governance innerhalb eines Unternehmens um das Low-Code-Regelwerk zu erweitern. Das Kapitel schließt mit einer Reflexion und Zusammenfassung der Ergebnisse, die als Grundlage für die nachfolgende Validierung in Kapitel 9 dienen (siehe Kapitel 8.3).

8.1 Gestaltungsmaßnahmen

In Kapitel 7 wurden Wirkbeziehungen zwischen den Low-Code Regeln und Anwendungsfalltypen identifiziert, sofern sich aus dem Merkmal oder der Ausprägung der Anwendungsfalltypen eine Maßnahme zum Einhalten der jeweiligen Low-Code Regel ergeben hat. Die so identifizierten Maßnahmen zur Einhaltung des Regelwerks gliedern sich in typenunabhängige und typenspezifische Maßnahmen. Abbildung 8-1 zeigt exemplarisch, wie aus den Wirkbeziehungen Maßnahmen identifiziert wurden (siehe A.12 und A.13). Die Zusammenhänge zwischen den Wirkbeziehungen und den Maßnahmen wurden auf Basis der in Kapitel 7 vorgestellten Expert*inneninterviews entwickelt (siehe A.5 bis A.11).

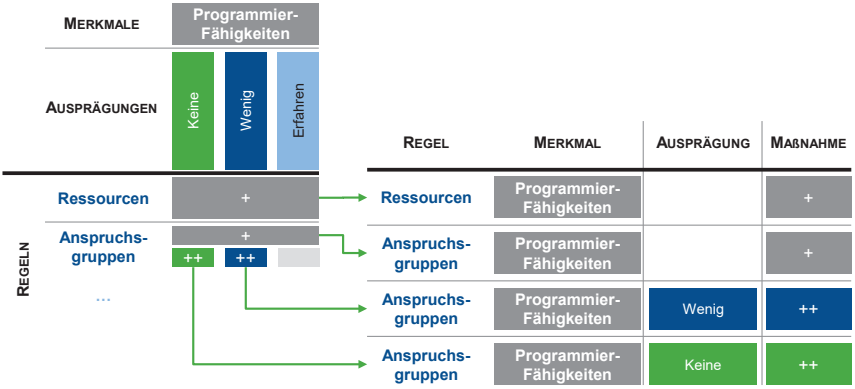

Abbildung 8-1: Exemplarisches Vorgehen zum Ableiten der Maßnahmen (eigene Darstellung)

Aus den Maßnahmen, die auf den individuellen Erfahrungen der Expert*innen beruhen, lassen sich nun typenunabhängig und typenspezifische Gestaltungsmaßnahmen ableiten. Die Ableitung von Gestaltungsmaßnahmen ermöglicht es, aus den spezifischen, praktischen Erfahrungen der interviewten Expert*innen allgemeine, übertragbare Prinzipien zu entwickeln. Eine detaillierte Auflistung aller typenunabhängigen und typenspezifischen Gestaltungsmaßnahmen ist im Anhang zu finden (siehe A.14). Die Gestaltungsmaßnahmen bieten einen Rahmen, der in verschiedenen Kontexten anwendungsfallspezifisch angewendet werden kann, um das in Kapitel 6 erarbeitete Low-Code-Regelwerk einzuhalten.

Nachfolgend werden zunächst die typenunabhängigen und anschließend die typenspezifischen Gestaltungsmaßnahmen im Detail vorgestellt. Die Anzahl an Gestaltungsmaßnahmen steigt mit der Komplexität und Wichtigkeit der Anwendungsfalltypen an. Folglich hat Typ I, Entwicklung durch Fachbereiche, die wenigsten Gestaltungsmaßnahmen und Typ III, Entwicklung durch IT-Spezialist*innen, die meisten Gestaltungsmaßnahmen. Da die Gestaltungsmaßnahmen dazu dienen, die in Kapitel 6 definierten Low-Code-Regeln einzuhalten, orientiert sich die Struktur der Gestaltungsmaßnahmen an der Struktur des Regelwerks.

8.1.1 Typenunabhängige Gestaltungsmaßnahmen

Die typenunabhängigen Gestaltungsmaßnahmen werden nacheinander anhand der fünf Dimensionen des Regelwerks „Vorgeben“, „Planen“, „Aufbauen“, „Betreiben“ und „Überwachen“ strukturiert vorgestellt (siehe Kapitel 6).

Vorgeben

Die Maßnahmen in der Dimension „Vorgeben“ zielen darauf ab, die Low-Code-Kompetenzen innerhalb der Organisation gezielt zu fördern und die Ressourcen im Bereich der Low-Code-Softwareentwicklung zu optimieren (siehe Abbildung 8-2).

Abbildung 8-2: Übersicht der typenunabhängigen Gestaltungsmaßnahmen in der Dimension „Vorgeben“ (eigene Darstellung)

Im Bereich der Ressourcen empfiehlt es sich, Low-Code-Schulungen zu erarbeiten, die sich nach den vorhandenen Programmierfähigkeiten der Mitarbeiter*innen und der Komplexität der geplanten Anwendungen richten. Ziel ist es, durch gezielte Bildungsmaßnahmen die Fähigkeit der Mitarbeiter, mit Low-Code-Plattformen zu arbeiten, zu verbessern, was die Entwicklung von Anwendungen beschleunigen und die Abhängigkeit von traditionellen Programmierfähigkeiten verringert.

Bei den Anspruchsgruppen wird die Einrichtung eines „Center of Excellence“ vorgeschlagen, um den Austausch und die kontinuierliche Verbesserung zu fördern. Ein solches Zentrum dient als Plattform für Mitarbeitende aus verschiedenen Bereichen, um Best Practices auszutauschen, Fähigkeiten zu entwickeln und zusammenzuarbeiten, um die Qualität und Effektivität der Programmierarbeit zu steigern.

Planen

Die Maßnahmen in der Dimension „Planen“ konzentrieren sich auf die Low-Code-IT-Architektur und die Qualität der Low-Code-Anwendungen. Dabei wird ein besonderer Fokus auf die Schnittstellen innerhalb der IT-Architektur gelegt (siehe Abbildung 8-3).

Abbildung 8-3: Übersicht der typenunabhängigen Gestaltungsmaßnahmen in der Dimension „Planen" (eigene Darstellung)

Für die IT-Architektur sollen Richtlinien definiert werden, die als Rahmenwerk für die Entwicklung und Wartung der IT-Systemlandschaft dienen, insbesondere der Schnittstellen zwischen den verschiedenen Komponenten. Es wird empfohlen, zu Beginn standardisierte Schnittstellen festzulegen. Die Standardisierung der Schnittstellen ist entscheidend, um eine nahtlose Kommunikation zwischen den unterschiedlichen IT-Systemen zu ermöglichen und die Integration von Drittsystemen zu vereinfachen. Die technische Integration der Low-Code-Plattform in die Technologiearchitektur und die Implementierung von standardisierten Schnittstellen sind zwei weitere technische

Maßnahmen, welche Konsistenz und Effizienz im operativen Betrieb gewährleisten und die praktische Umsetzung der festgelegten Standards umfassen.

Im Bereich der Qualität wird die Einführung einer Feedbackschleife mit Anwender*innen vor Inbetriebnahme der Anwendung als Maßnahme genannt. Die Einbeziehung von Nutzerfeedback vor dem offiziellen Start der Anwendung kann dabei helfen, die Anwendungsfunktionalität zu verbessern, mögliche Risiken zu identifizieren und sicherzustellen, dass die Anwendung den Anforderungen der Nutzer entspricht.

Aufbauen

In der Dimension „Aufbauen“ sind Maßnahmen für die Phasen der Anforderungsdefinition und der Lösungserstellung aufgeführt. Die Maßnahmen sollen dabei helfen, die Entwicklung von Low-Code-Anwendungen strukturiert und benutzerzentriert zu gestalten und die Qualität und Effizienz im Entwicklungsprozess zu verbessern (siehe Abbildung 8-4).

AUFBAUEN

Organisatorisch	Technisch
Anforderungsdefinition • Kriterienkatalog zur Anforderungsdefinition und Überprüfung der Low-Code-Eignung erstellen • Richtlinien zur Überprüfung der notwendigen Programmierfähigkeiten zur Anwendungsentwicklung einführen • Übersicht über integrierbare Daten erstellen, um Datenbedarf der Anwendung abgleichen zu können	
Lösungserstellung • Standards unter Berücksichtigung aller Nutzer*innen einführen • Modellierungsrichtlinien definieren • Lösungserstellung durch erfahrene Low-Code Entwickler*innen überprüfen lassen	• Designsystem mit Standard-UI-Komponenten implementieren

Abbildung 8-4: Übersicht der typenunabhängigen Gestaltungsmaßnahmen in der Dimension „Aufbauen" (eigene Darstellung)

Für die Anforderungsdefinition soll ein Kriterienkatalog zur Überprüfung der Low-Code-Eignung erstellt werden. Dieser Katalog dient dazu, zu bestimmen, welche Anwendungen für Low-Code-Entwicklungen geeignet sind. Es werden Richtlinien zur Überprüfung der notwendigen Programmierfähigkeiten zur Anwendungsentwicklung eingeführt. Ziel ist es, sicherzustellen, dass die Entwickelnden über die erforderlichen Kenntnisse verfügen die entsprechenden Anwendungen zu entwickeln. Es ist wichtig, eine Übersicht über integrierbare Daten zu erstellen, um den Datenbedarf der Anwendung abgleichen zu können. Dies unterstützt die Entscheidung, welche Datenquellen und -strukturen in die Anwendung einfließen sollen.

Bei der Lösungserstellung müssen Modellierungsrichtlinien definiert werden, die als Leitfaden für die Gestaltung der Low-Code-Anwendung dienen. Die Lösungserstellung sollte durch erfahrene Low-Code-Entwickelnde überprüft werden. Diese Qualitätssicherungsmaßnahme gewährleistet, dass die entwickelten Lösungen den Standards und Anforderungen entsprechen. Es gilt, Standards unter Berücksichtigung aller Nutzer einzuführen. Diese Standards sollen die Nutzbarkeit und Zugänglichkeit der Anwendungen sicherstellen. Schließlich soll ein Designsystem mit Standard-UI-

Komponenten implementiert werden. Dies fördert die Konsistenz und Effizienz im Designprozess und sorgt für eine einheitliche Benutzererfahrung.

Ausführen

In der Dimension „Ausführen“ werden Maßnahmen für den Betrieb und die Handhabung von Support aufgeführt. Die Maßnahmen zielen darauf ab, den Betrieb von Anwendungen zu optimieren und einen reibungslosen, effektiven Supportprozess für die Anwender*innen zu etablieren (siehe Abbildung 8-5).

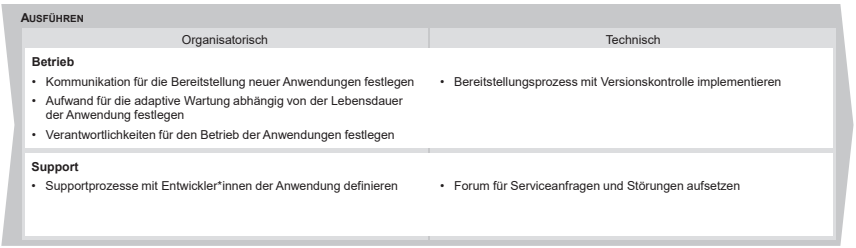

Abbildung 8-5: Übersicht der typenunabhängigen Gestaltungsmaßnahmen in der Dimension „Ausführen" (eigene Darstellung)

Im Betrieb wird festgelegt, wie die Kommunikation für die Bereitstellung neuer Anwendungen gestaltet werden soll, um die Wichtigkeit und den Einfluss auf die Organisation zu berücksichtigen. Der Aufwand für die adaptive Wartung wird abhängig von der Lebensdauer der Anwendung festgelegt, was eine planvolle Ressourcenallokation ermöglicht. Die Verantwortlichkeiten für den Betrieb der Anwendungen werden klar definiert, sodass jeder Beteiligte seine spezifischen Rollen und Pflichten kennt. Ein Bereitstellungsprozess mit Versionskontrolle wird implementiert, um sicherzustellen, dass Änderungen nachvollziehbar sind und die Stabilität kritischer Systeme gewährleistet ist.

Für Support definiert man Supportprozesse in Zusammenarbeit mit den Entwicklern der Anwendung. In den Supportprozessen wird festgelegt, wer für welche Wartungsaufgabe, Entwicklungsunterstützung und Fehlermeldung verantwortlich ist und welche Aufgaben welche Priorität haben. Dies gewährleistet, dass das Fachwissen der Entwickelnden genutzt wird, um einen effizienten und wirksamen Support zu etablieren. Zusätzlich wird ein Forum aufgesetzt, das als Plattform dient, um Wissen auszutauschen und Unterstützung bei Problemen anzubieten, die spezielle Programmierfähigkeiten erfordern.

Überwachen

In der Dimension „Überwachen“ werden Richtlinien und Maßnahmen in Bezug auf Datenmanagement, Sicherheitsservices und Wissensmanagement aufgeführt. Die aufgeführten Maßnahmen tragen dazu bei, das Daten- und Sicherheitsmanagement sowie das Wissensmanagement im Unternehmenskontext zu strukturieren und zu optimieren (siehe Abbildung 8-6).

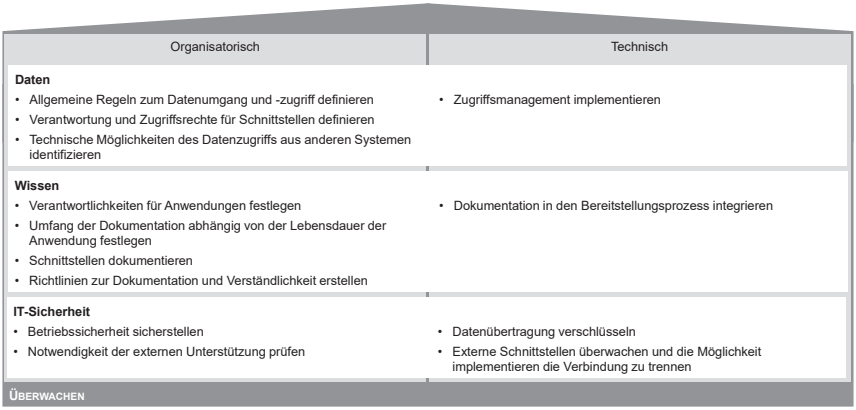

Abbildung 8-6: Übersicht der typenunabhängigen Gestaltungsmaßnahmen in der Dimension „Überwachen" (eigene Darstellung)

Für das Datenmanagement sind allgemeine Regeln zum Datenumgang und -zugriff zu definieren, um den sicheren und effizienten Umgang mit Daten im Unternehmen zu gewährleisten. Es ist erforderlich, Verantwortung und Zugriffsrechte für Schnittstellen zu definieren. Das gewährleistet klare Zuständigkeiten und eine sichere Datenübertragung zwischen den Systemen. Um die technischen Möglichkeiten des Datenzugriffs aus anderen Systemen zu identifizieren, müssen die notwendigen Programmierfähigkeiten berücksichtigt werden, was eine Basis für die Integration unterschiedlicher Systemlandschaften bietet. Das Zugriffsmanagement für Schnittstellen soll implementiert werden, um die Kontrolle über den Datenfluss zwischen verschiedenen Diensten und Anwendungen zu verbessern.

Um Wissen über die Low-Code-Anwendungen zu gewährleisten, müssen Verantwortlichkeiten für Anwendungen festgelegt werden, was die Nutzbarkeit und Wartung der Anwendungen vereinfacht. Der Umfang der Dokumentation soll abhängig von der Lebensdauer der Anwendung festgelegt werden, was zur Nachhaltigkeit und Wartbarkeit von Anwendungen beiträgt. Die Schnittstellen müssen dokumentiert werden, um ein klares Verständnis über die Datenflüsse und Abhängigkeiten zu schaffen. Die Dokumentation ist in den Bereitstellungsprozess zu integrieren, um sicherzustellen, dass die Anwendungskontexte und Verfahren jederzeit nachvollziehbar sind.

Für die IT-Sicherheit ist die Betriebssicherheit sicherzustellen, indem die Low-Code-Entwickelnden zur Aufrechterhaltung und Verbesserung der IT-Sicherheit geschult werden. Es ist zu prüfen, ob die Notwendigkeit der externen Unterstützung besteht, um Sicherheitsdienste zu verstärken und Lücken zu schließen. Es wird angestrebt, die externen Schnittstellen zu überwachen und bei Bedarf die Verbindung zu kappen, was als präventive Sicherheitsmaßnahme dient. Für alle Schnittstellen soll die Datenübertragung verschlüsselt werden, um die Integrität und Vertraulichkeit der Daten zu gewährleisten.

Nachdem zunächst die typenunabhängigen Gestaltungsmaßnahmen präsentiert wurden, folgen im Anschluss die Gestaltungsmaßnahmen für Anwendungsfalltyp I, II und III.

8.1.2 Gestaltungsmaßnahmen: Typ I

Bei Anwendungsfalltyp I handelt es sich um die Entwicklung der Low-Code-Anwendungen durch den Fachbereich. Die spezifischen Gestaltungsmaßnahmen für Typ I konzentrieren sich auf den Austausch der Anspruchsgruppen, die Anforderungsdefinition und das Wissensmanagement und sind in Abbildung 8-7 dargestellt.

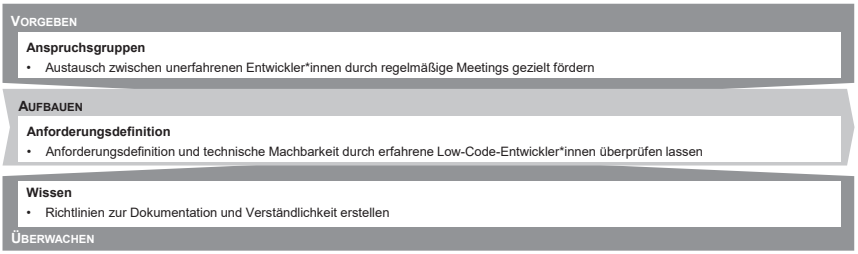

Abbildung 8-7: Spezifische Gestaltungsmaßnahmen für Typ I (eigene Darstellung)

Für Anspruchsgruppen mit wenig oder keinen Programmierfähigkeiten soll der Austausch zwischen unerfahrenen Entwickelnden durch regelmäßige Meetings gefördert werden. Das Ziel ist es, eine Lernkultur zu schaffen und den Wissensaustausch zu erleichtern.

Bei der Anforderungsdefinition wird empfohlen, dass die definierten Anforderungen durch erfahrene Low-Code-Entwickelnde auf die technische Machbarkeit geprüft werden sollten. Dies stellt sicher, dass alle Lösungen praktikabel und speziell auf Low-Code-Entwicklungsprozesse abgestimmt sind. Die Überprüfung der Anforderungsdefinition ist besonders wichtig, wenn die notwendigen Programmierfähigkeiten entweder nicht vorhanden oder nur in geringem Umfang vorhanden sind.

Für das Wissensmanagement sollen Richtlinien zur Dokumentation und Verständlichkeit erstellt werden. Dies kann dabei helfen, den Wissenstransfer innerhalb des Unternehmens zu verbessern und sicherzustellen, dass alle Beteiligten unabhängig von ihren Programmierfähigkeiten die Dokumentation verstehen und nutzen können.

Diese Maßnahmen unterstreichen die Wichtigkeit der Einbeziehung erfahrener Low-Code-Entwickelnder in den Entwicklungsprozess, sowohl bei der Anforderungsdefinition als auch bei der technischen Machbarkeitsprüfung. Darüber hinaus wird die Bedeutung von Wissensaustausch und klaren Dokumentationsrichtlinien hervorgehoben, um eine inklusive und lernfördernde Umgebung zu schaffen.

8.1.3 Gestaltungsmaßnahmen: Typ II

Anwendungsfalltyp II beschreibt die Entwicklung von Low-Code-Anwendungen durch Fachbereich und IT gemeinsam. Die spezifischen Gestaltungsmaßnahmen für Typ II sollten die Effizienz und Qualität in verschiedenen Bereichen der Low-Code-Softwareentwicklung in der Zusammenarbeit zwischen Fachbereich und IT-Spezialist*innen steigern (siehe Abbildung 8-8).

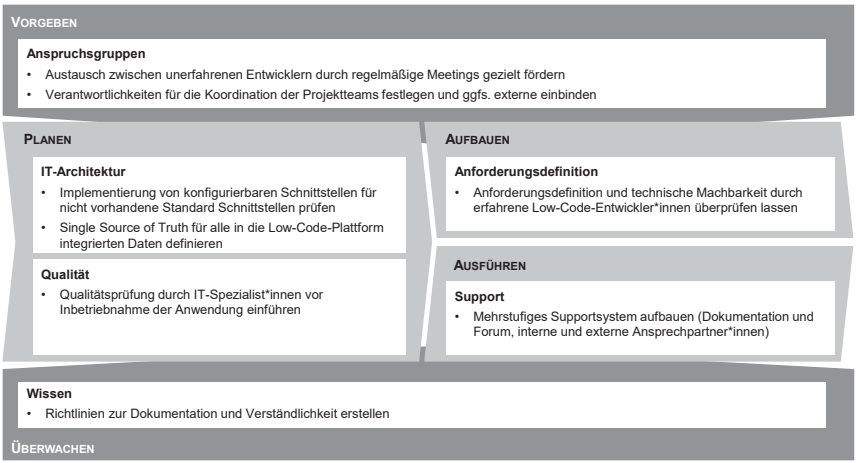

Abbildung 8-8: Spezifische Gestaltungsmaßnahmen für Typ II (eigene Darstellung)

Ähnlich wie bei Typ I soll der Austausch zwischen den verschiedenen Anspruchsgruppen durch regelmäßige Meetings gefördert werden. Dabei liegt der Schwerpunkt insbesondere auf dem Austausch zwischen erfahrenen IT-Spezialist*innen und den in der Softwareentwicklung weniger erfahrenen Fachbereichen. Zudem sollen die Verantwortlichkeiten für die Koordination der Projektteams klar definiert werden, wobei gegebenenfalls Unterstützung durch externe IT-Spezialist*innen einbezogen werden kann.

In der IT-Architektur geht es darum, eine „Single Source of Truth" für alle in die Low-Code-Plattform integrierten Daten zu definieren und die Implementierung von konfigurierbaren Schnittstellen zu prüfen, um Flexibilität bei fehlenden Standard-Schnittstellen zu ermöglichen. Im Bereich Qualität wird die Einführung einer Qualitätsprüfung durch IT-Spezialist*innen vor der Inbetriebnahme von Anwendungen vorgesehen, um sicherzustellen, dass alle Releases den hohen Anforderungen entsprechen.

Für die Anforderungsdefinition ist vorgesehen, sowohl die Anforderungen als auch die technische Machbarkeit von erfahrenen Low-Code-Entwickelnden, und in einem weiteren Schritt auch von Endnutzern, überprüfen zu lassen.

Im Support ist der Aufbau eines mehrstufigen Supportsystems geplant, das Dokumentationen und Foren sowie interne und externe Ansprechpartner umfasst, um ein umfassendes Supportnetzwerk zu schaffen.

Schließlich sollen im Bereich Wissen Richtlinien zur Dokumentation und Verständlichkeit erstellt werden, die sicherstellen, dass alle relevanten Informationen zugänglich und verständlich sind, um die Wissensvermittlung innerhalb der Organisation zu verbessern.

8.1.4 Gestaltungsmaßnahmen: Typ III

Bei Anwendungsfalltyp III handelt es sich um die Entwicklung der Low-Code-Anwendungen durch die IT. Es werden Maßnahmen aufgeführt, die verschiedene Aspekte der Low-Code-Entwicklung und Anwendungs-Verwaltung adressieren. Diese Maßnahmen decken die Regeln für Anspruchsgruppen, Ressourcen, IT-Architektur, Qualität, Anforderung, Lösungserstellung, Betrieb, Support, Datenmanagement, Sicherheit und Wissensmanagement ab (siehe Abbildung 8-9).

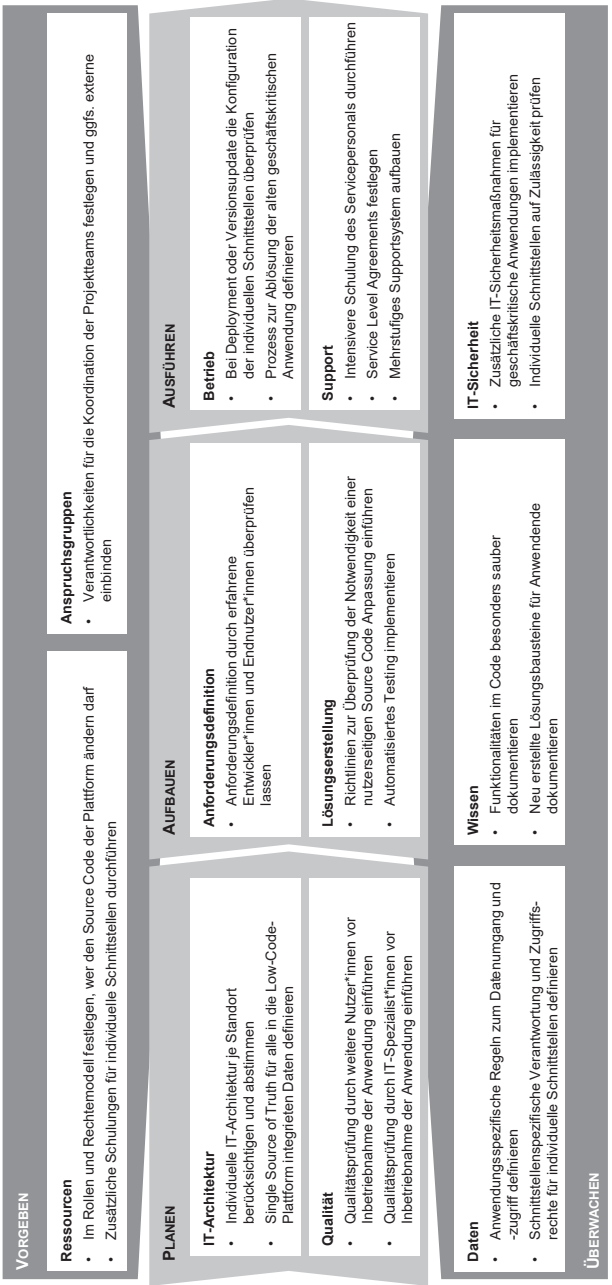

Abbildung 8-9: Spezifische Gestaltungsmaßnahmen für Typ III (eigene Darstellung)

Für die Anspruchsgruppen wird empfohlen, Verantwortlichkeiten für die Koordination von Projektteams zu definieren und bei Bedarf externe Expertise hinzuzuziehen. In Bezug auf die zur Verfügung stehenden Ressourcen sind Festlegungen im Rollen- und Rechtemodell vorgesehen, um zu bestimmen, wer Änderungen am Source Code vornehmen darf, und es sind zusätzliche Schulungen für Source Code-Anpassungen und individuelle Schnittstellen geplant.

Die IT-Architektur soll individuell auf die Standorte abgestimmt sein, und es soll eine „Single Source of Truth" für alle in die Low-Code-Plattform integrierten Daten definiert werden. Qualitätsprüfungen sollen sowohl durch IT-Spezialist*innen als auch durch Nutzer*innen vor der Inbetriebnahme einer Anwendung stattfinden. Da die Anwendungen des Typs III als geschäftskritisch eingestuft werden, ist zusätzlich automatisiertes Testing vorgesehen, um die Anwendungen permanent überwachen zu können und so Fehler und Ausfälle zu vermeiden.

Anforderungen sollen durch erfahrene Entwickelnde und Endnutzer*innen geprüft werden. Richtlinien zur Lösungserstellung sollen etabliert werden, die die Notwendigkeit von nutzerseitigen Source-Code-Anpassungen überprüfen.

Für den Betrieb wird ein Prozess zur Ablösung alter geschäftskritischer Anwendungen benötigt und bei jedem Deployment oder Update sollen individuelle Code-Bausteine und Schnittstellen überprüft werden. Der Aufbau eines mehrstufigen Supportsystems ist geplant, Service Level Agreements sollen festgelegt und das Servicepersonal soll intensiver geschult werden.

Anwendungsspezifische Regeln für den Datenumgang sollen definiert und Verantwortlichkeiten für individuelle Schnittstellen festgelegt werden. Individuelle Schnittstellen sind auf Zulässigkeit zu prüfen. Für geschäftskritische Anwendungen sollten zusätzliche Sicherheitsservices wie Penetration Testing implementiert werden. Die Dokumentation soll auch von Dritten geprüft werden, Funktionen im Code sind sauber zu dokumentieren und neu erstellte Lösungsbausteine sollen für Anwender*innen dokumentiert werden.

8.2 Anwendung des Regelwerks

Nachdem im vorherigen Kapitel die Gestaltungsmaßnahmen zur Einhaltung des Regelwerks vorgestellt wurden, soll nun ein Vorgehen entwickelt werden, wie das Low-Code-Regelwerk mithilfe der vier zuvor entwickelten Modelle in einem ganzheitlichen Ansatz angewendet werden kann. Dazu wird zunächst der Anwendungsraum des Vorgehens spezifiziert (siehe Kapitel 8.2.1) und anschließend das Vorgehen zur praktischen Anwendung des Regelwerks vorgestellt (siehe Kapitel 8.2.2).

8.2.1 Spezifizierung des Anwendungsraums

In diesem Abschnitt wird der Anwendungsbereich des Regelwerks detaillierter betrachtet. Diese Vertiefung ist notwendig, da die Implementierung von Low-Code in Unternehmen je nach Stadium und Reifegrad variieren kann. Deshalb werden zunächst die

unterschiedlichen Implementierungsformen erörtert und hinsichtlich ihrer Kompatibilität mit dem Regelwerk bewertet. Abbildung 8-10 stellt vier verschiedene Implementierungsansätze für softwarebasierte Technologien dar.

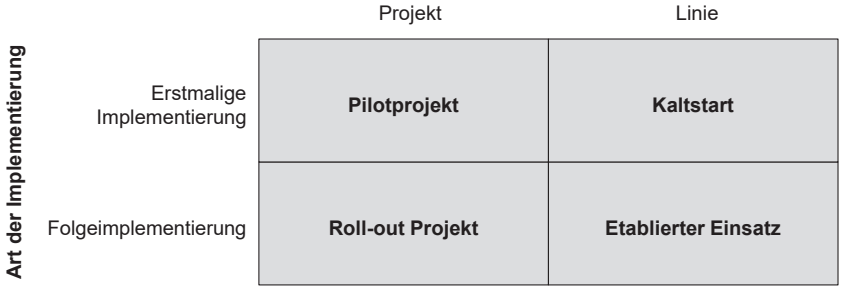

Abbildung 8-10: Implementierungsansätze für softwarebasierte Technologien (s. SMEETS ET AL. 2019, S. 58)

Gemäß SMEETS ET AL. können diese Implementierungsformen in zwei Hauptkategorien eingeteilt werden: Die erstmalige Implementierung und die Folgeimplementierung. Jede dieser Kategorien kann entweder als Projekt oder direkt im operativen Geschäft (Linie) realisiert werden. Die Autor*innen weisen darauf hin, dass es zwar weitere Formen der Implementierung geben könnte, diese aber häufig Variationen der in Abbildung 8-10 dargestellten Hauptformen sind und daher in ihrer Untersuchung nicht weiter berücksichtigt werden (vgl. SMEETS ET AL. 2019, S. 59). Die Ergebnisse der geführten Expert*inneninterviews zeigen auf, dass die Erstimplementierung fast immer als Projekt realisiert wird. Dies liegt hauptsächlich am anfänglichen Mangel an technologischem Know-how. Auch bei den nachfolgenden Implementierungen wird oft noch die Projektform gewählt. Der Grund hierfür ist, dass der Übergang zum regulären Linienbetrieb in der Regel mehr Vorbereitungszeit erfordert, als das erste Implementierungsprojekt zur Verfügung stellen kann. Langfristig streben Unternehmen jedoch meistens an, Folgeimplementierungen in den regulären Linienbetrieb zu überführen (s. SMEETS ET AL. 2019, S. 60).

Weiter unterscheidet MURDOCH zwischen drei Phasen von Robotic Process Automation (RPA) in einer Organisation, welche auf Low-Code übertragen werden können (s. MURDOCH 2018):

- **Initialisierungsphase:** In der Initialisierungsphase beginnen Organisationen mit dem Einsatz von Low-Code-Plattformen oder haben diesen Schritt kürzlich vollzogen. In dieser Phase wird Low-Code vorrangig für kleinere und weniger komplexe Anwendungen eingesetzt. Dies dient dazu, grundlegende Erfahrungen zu sammeln und die Technologie besser zu verstehen. Ein wesentlicher Fokus liegt auf der Entwicklung einer Low-Code-Strategie und der Etablierung einer Governance-Struktur.
- **Industrialisierungsphase:** Die Industrialisierungsphase kennzeichnet eine Ausweitung der Low-Code-Nutzung. Low-Code wird nun für komplexere

Anwendungen eingesetzt und integriert sich stärker in die bestehende IT-Landschaft der Organisation. In dieser Phase finden sich Low-Code-Projekte zunehmend in verschiedenen Geschäftsbereichen, oft unter der Leitung spezialisierter Teams oder Arbeitsgruppen. Low-Code wird zu einem festen Bestandteil der IT-Strategie der Organisation, wobei klare Richtlinien und Standards für die Entwicklung und den Einsatz festgelegt werden können.

- **Institutionalisierungsphase:** In der Institutionalisierungsphase schließlich erreicht die Integration von Low-Code in die Unternehmensprozesse eine neue Ebene. Low-Code-Anwendungen erweitern ihre Anwendungsbereiche weit über die üblichen Grenzen hinaus und werden tief in die Betriebsabläufe integriert. Die Nutzung von Low-Code ist fest in der Unternehmenskultur verankert, und das erforderliche Wissen ist unter den Mitarbeitenden weit verbreitet. In dieser Phase kommen fortgeschrittene Kennzahlen und Analysetools zum Einsatz, um die Leistung und den Return on Investment der Low-Code-Anwendungen kontinuierlich zu messen und zu optimieren. Dadurch wird sichergestellt, dass die Organisation das volle Potenzial von Low-Code ausschöpft und stetig an die sich ändernden Anforderungen und Möglichkeiten anpasst.

Das im Anschluss vorgestellte Vorgehen zur Anwendung des Regelwerks sollte den Anspruch erfüllen, für alle drei Phasen der Low-Code-Implementierung – Initialisierung, Umsetzung und Skalierung – relevant und nützlich zu sein. Das Vorgehen resultiert in einer Liste an ausgewählten Gestaltungsmaßnahmen, die Unternehmen in ihre bestehende IT-Governance integrieren können. Die abgeleiteten Gestaltungsmaßnahmen sollen Unternehmen dabei unterstützen, die Vorgaben aus dem Low-Code-Regelwerk zu erfüllen. Einige Gestaltungsmaßnahmen sind besonders in der Initialisierungsphase wichtig, während andere bei der Entwicklung jeder neuen Low-Code-Anwendung oder kontinuierlich beachtet werden müssen. Durch Integration in die bestehende IT-Governance können Organisationen sicherstellen, dass die Einführung und Integration von Low-Code-Anwendungen in ihre vorhandenen Strukturen und Abläufe effizient und wirksam gestaltet werden. Es ist wichtig zu verstehen, dass die vollständige Umsetzung aller Gestaltungsmaßnahmen einen Idealzustand darstellt, den Unternehmen schrittweise erreichen können.

8.2.2 Vorgehen zur Anwendung des Regelwerks

Das Vorgehen zur Anwendung des Regelwerks basiert auf vier Schritten und integriert die vier zuvor erarbeiteten Modelle in einen Gesamtkontext (siehe Abbildung 8-11). Das Vorgehen wird im Folgenden Schritt für Schritt erläutert.

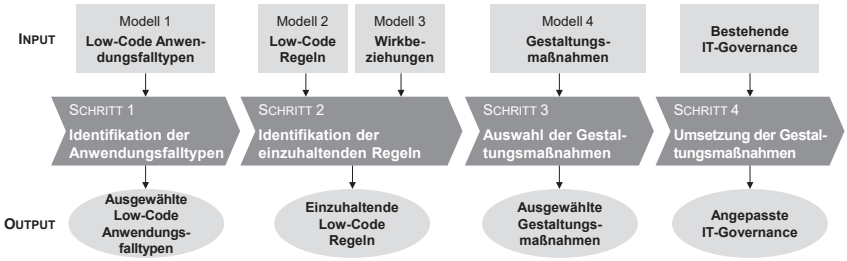

Abbildung 8-11: Vorgehen zur Anwendung des Regelwerks (eigene Darstellung)

Schritt 1: Identifikation der Anwendungsfalltypen

Der erste Schritt besteht in der Identifikation der geplanten und bereits implementierten Anwendungsfalltypen. Als Input dient das erste Modell (siehe Kapitel 5), welches die Identifikation und Typisierung verschiedener Low-Code-Anwendungsfälle erleichtert, indem es Unternehmen durch einen entwickelten morphologischen Kasten ermöglicht, ein umfassendes Verständnis für verschiedene Anwendungsfalltypen von Low-Code zu entwickeln. Während dieses Prozessschritts werden die spezifischen Anwendungsfalltypen ausgewählt, die entweder schon implementiert oder für die zukünftige Umsetzung geplant sind. Das Ergebnis ist eine Liste der ausgewählten Low-Code-Anwendungsfalltypen, die als Grundlage für die weiteren Schritte dient.

Schritt 2: Identifikation der einzuhaltenden Regeln

Im nächsten Schritt erfolgt die Identifikation der einzuhaltenden Regeln für die ausgewählten Anwendungsfalltypen. Hierbei werden das zweite Modell und das dritte Modell als Input verwendet. Das zweite Modell konzentriert sich auf die Identifikation von Regeln, die den mit Low-Code einhergehenden Risiken entgegenwirken. Es bietet Unternehmen einen ganzheitlichen Überblick über Dinge, die bei der Einführung und dem Betrieb von Low-Code beachtet werden müssen. Das dritte Modell verdeutlicht die Wirkbeziehungen, die beim Einsatz des ausgewählten Low-Code-Anwendungsfalltyps beachtet werden müssen. Der Prozess umfasst die Schaffung eines klaren Verständnisses darüber, welche spezifischen Regeln für jeden Anwendungsfalltyp eingehalten werden müssen. Dies beinhaltet eine gründliche Analyse des Regelwerks und der Wirkzusammenhänge, um sicherzustellen, dass alle relevanten Regeln für jeden Anwendungsfalltyp klar definiert und nachvollziehbar sind. Das Ergebnis ist ein eingegrenztes Regelwerk, das auf die Besonderheiten jedes ausgewählten Anwendungsfalltyps zugeschnitten ist.

Schritt 3: Auswahl der Gestaltungsmaßnahmen

Nachdem die einzuhaltenden Regeln für die jeweiligen Anwendungsfalltypen identifiziert wurden, erfolgt die Auswahl der entsprechenden Gestaltungsmaßnahmen. Als Input dienen hierbei die ausgewählten Anwendungsfalltypen, das eingegrenzte Regelwerk für jeden Anwendungsfalltyp und das vierte Modell. Das vierte Modell schlägt typenspezifische Gestaltungsmaßnahmen zur Einhaltung des Low-Code-Regelwerks

vor. In diesem Prozessschritt wird für jeden Anwendungsfalltyp ein tiefgehendes Verständnis für die einzelnen Wirkzusammenhänge geschaffen. Auf Basis dieses Verständnisses werden spezifische Gestaltungsmaßnahmen identifiziert, die zur Einhaltung der Regeln beitragen und die Effektivität der Anwendungsfalltypen maximieren. Das Ergebnis ist eine umfassende Liste ausgewählter Gestaltungsmaßnahmen, die gezielt auf die Bedürfnisse und Anforderungen der einzelnen Anwendungsfalltypen abgestimmt sind und für die Einhaltung des Low-Code-Regelwerks sorgen.

Schritt 4: Umsetzung der Gestaltungsmaßnahmen

Der abschließende Schritt im Prozess ist die Umsetzung der identifizierten Gestaltungsmaßnahmen. Dabei dienen die ausgewählten Gestaltungsmaßnahmen als Input. Dieser Prozess umfasst die Strukturierung der Gestaltungsmaßnahmen und deren Integration in die bestehende IT-Governance des Unternehmens. Hierbei wird darauf geachtet, dass die Maßnahmen nahtlos in die bestehenden Strukturen und Prozesse eingebettet werden, um eine effiziente und effektive Umsetzung zu gewährleisten. Das Ergebnis dieses Schrittes ist eine angepasste IT-Governance, die speziell auf die Anforderungen und Besonderheiten der Low-Code-Anwendungen zugeschnitten ist und die Einhaltung des Regelwerks sicherstellt.

8.3 Reflexion und Zusammenfassung der Ergebnisse

In den vorherigen Unterkapiteln wurde untersucht, mit welchen Gestaltungsmaßnahmen das in Kapitel 6 beschriebene Regelwerk anwendungsfallbasiert eingehalten werden kann. Zunächst wurden die typenunabhängigen und typenspezifischen Gestaltungsmaßnahmen zur Einhaltung des Regelwerks vorgestellt (siehe Kapitel 8.1). Anschließend wurde ein Vorgehen zur Anwendung des Regelwerks in der Praxis beschrieben. Als Anwendungsraum eignen sich sowohl die erstmalige Low-Code-Implementierung in der Initialisierungsphase als auch nachfolgende Implementierungen in der Industrialisierungs- oder Institutionalisierungsphase (siehe Kapitel 8.2).

Die Gliederung des Vorgehens in vier Schritte bietet eine klare Struktur, die Unternehmen dabei unterstützt, das Low-Code-Regelwerk anzuwenden. Beginnend mit der Identifikation der relevanten Anwendungsfalltypen (1), über die Bestimmung der einzuhaltenden Regeln (2) und die Auswahl passender Gestaltungsmaßnahmen (3), bis hin zur Implementierung dieser Maßnahmen in die bestehende IT-Governance-Struktur (4), stellt dieser Prozess ein umfassendes und systematisches Vorgehen zur Anpassung der IT-Governance dar. Insbesondere die Unterscheidung zwischen typenunabhängigen und typenspezifischen Gestaltungsmaßnahmen stellt sicher, dass sowohl allgemeine als auch spezifische Anforderungen berücksichtigt werden. Dies trägt dazu bei, dass die IT-Governance nicht nur den allgemeinen Standards entspricht, sondern auch flexibel genug ist, um die speziellen Anforderungen der einzelnen Anwendungsfalltypen zu erfüllen. Die vorgeschlagenen Gestaltungsmaßnahmen wurden so erarbeitet, dass sie sowohl für eine erstmalige Implementierung als auch für kontinuierliche Low-Code-Implementierungen geeignet sind. Dies ermöglicht Unternehmen, schrittweise Erfahrungen zu sammeln und ihre Governance-Struktur

kontinuierlich zu verbessern. Durch das vorgeschlagene Vorgehen wird sichergestellt, dass die IT-Governance des Unternehmens mithilfe des Regelwerks und der Gestaltungsmaßnahmen effektiv auf die Integration von Low-Code-Anwendungen vorbereitet ist und kontinuierlich verbessert werden kann.

Somit wurde die vierte untergeordnete und handlungsleitende Forschungsfrage beantwortet:

Wie kann das Vorgehen zur anwendungsfallbasierten Definition eines Regelwerks für Low-Code in produzierenden Unternehmen gestaltet werden?

Die Validierung der Ergebnisse im nächsten Kapitel wird entscheidend sein, um die Wirksamkeit der vorgeschlagenen Maßnahmen zu bestätigen und gegebenenfalls Anpassungen vorzunehmen.

9 Validierung des Regelwerks in der betrieblichen Praxis

Kapitel 9 konzentriert sich auf die Überprüfung des entwickelten Regelwerks im praktischen Anwendungskontext. Entsprechend der von ULRICH beschriebenen Forschungskonzeption und dem Ziel der Handlungswissenschaften, praxisrelevantes Wissen zu generieren, liegt der Fokus dieses Kapitels auf der Anwendbarkeit der Ergebnisse in der Praxis und nicht auf der theoretischen Validierung (s. ULRICH 1984, S. 7).

In Kapitel 9.1 wird zunächst das Vorgehen bei der exemplarischen Anwendung des entwickelten Regelwerks erläutert. Die Validierung der Ergebnisse erfolgt anhand von zwei konkreten Fallbeispielen aus der betrieblichen Praxis. In diesen Beispielen werden die einzelnen Modelle angewendet und gemäß den in Kapitel 4.1 definierten formal-konzeptionellen und inhaltlichen Anforderungen bewertet (siehe Kapitel 9.2 und 9.3). Dadurch werden sowohl die Einhaltung der Grundsätze ordnungsgemäßer Modellierung als auch der wissensgenerierende Mehrwert für die Praxis sichergestellt. Mit der Anwendung und Überprüfung des Regelwerks im spezifischen unternehmerischen Anwendungskontext kann der Prozess der angewandten Forschung nach ULRICH als abgeschlossen betrachtet werden (s. ULRICH 1984, S. 193).

9.1 Vorgehensweise der Validierung

Für die Validierung des entwickelten Regelwerks wurden Vertreter der folgenden zwei Unternehmen interviewt: Zentis Fruchtwelt GmbH & Co. KG und ein mittelständischer Automobilzulieferer. Durch die Interviews werden eine industrieseitige Validierung der entwickelten Modelle sowie die Anwendbarkeit des gesamten Regelwerks gewährleistet.

Zu Beginn der Interviews wurde die Notwendigkeit der Thematik verdeutlicht und die Historie des Ideenfindungsprozesses für das Dissertationsvorhaben geschildert. Anschließend stellten die Interviewpartner ihre Fallbeispiele vor, in denen die Unternehmen ihre individuellen Ausgangssituationen und Herausforderungen bei der Implementierung von Low-Code-Anwendungen darlegten. Bereits zu diesem Zeitpunkt konnte die Notwendigkeit eines ganzheitlichen Regelwerks herausgestellt werden.

Im weiteren Verlauf der Validierungsgespräche wurden die drei Anwendungsfalltypen durch die Autorin vorgestellt und anhand der Fallbeispiele auf ihre reale Existenz geprüft. Darauf aufbauend wurde das entwickelte Regelwerk mit den Interviewpartnern diskutiert und dessen Anwendbarkeit in der Praxis anhand des implementierten Anwendungsfalltyps validiert. Besonders wurden die Wirkungsbeziehungen zwischen den Typen und dem Regelwerk auf ihren Nutzen für die Praxis bewertet. Hierbei bestätigte sich die Annahme der Autorin, dass fehlende Kenntnisse über diese Wirkbeziehungen ein wesentlicher Faktor für die zahlreichen Herausforderungen in der Industrie sind. Nach der Vorstellung der Modelle wurde das Gestaltungsmodell, welches

Gestaltungsmaßnahmen für den Einsatz von Low-Code-Anwendungen enthält, mithilfe der Experten validiert. Abschließend erfolgte eine schrittweise Überprüfung der in Kapitel 4.1 entwickelten formal-konzeptionellen und inhaltlichen Anforderungen mit den Industrievertretern. Abbildung 9-1 bietet einen Überblick über die Fragestellungen, die zur Überprüfung der Anforderungen herangezogen wurden.

Formale Anforderungen	
Validität	Sind die Modelle interdisziplinär verständlich und auf ähnliche Probleme übertragbar?
Reliabilität	Führen die Modelle bei wiederholter Anwendung unter gleichen Bedingungen zu gleichen Ergebnissen?
Utilität	Sind die Modelle nützlich, verständlich und für die praktische Anwendbarkeit definiert?
Inhaltliche Anforderungen	
Merkmale und Ausprägungen	Sind die Merkmale und Ausprägungen vollständig, praxistauglich und überschneidungsfrei?
Heterogenität der Typen	Können die identifizierten Typen deutlich voneinander abgegrenzt werden?
Reale Existenz der Typen	Exisiteren die Typen in der Praxis und sind die Merkmalskombinationen logisch kohärent, empirisch überprüfbar und praktisch anwendbar?
Regelwerk	Berücksichtigt das Regelwerk alle relevanten und tatsächlich existenten Risiken von Low-Code?
Wirkbeziehungen	Wurde die wechselseitige Wirkbeziehung der Low-Code Anwendungsfalltypen und Regeln verständlich erklärt?
Gestaltungsmaßnahmen	Liefert das Regelwerk sowie die anwendungsfallspezifischen Gestaltungsmaßnahmen einen Mehrwert in der Praxis?
Innere Konsistenz	Stellt die Gesamtheit der Teilmodelle ein kohärentes Bild der komplexen Realität dar?
Visualisierung	Trägt die Visualisierung dazu bei, die Modelle klar und übersichtlich zu vermitteln?
Anpassungsfähigkeit	Sind die Modelle so gestaltet, dass sie vor dem Hintergrund eines technologischen Wandels anpassungsfähig sind?

Abbildung 9-1: Bewertungslogik zur Validierung formal-konzeptioneller und inhaltlicher Anforderungen (eigene Darstellung)

9.2 Fallbeispiel: Zentis Fruchtwelt GmbH & Co. KG

Im Zuge der Validierung wurde eine erste Diskussion der Ergebnisse mit einem Experten der Zentis Fruchtwelt GmbH & Co. KG durchgeführt. Der Ansprechpartner, Head of Business Process Optimization & Data, verfügt über umfassende Kenntnisse in der Implementierung von Low-Code-Anwendungen. Das Gespräch fand virtuell per Videotelefonat statt. Zunächst lag der Fokus der Validierung auf der Vorstellung der im Rahmen der Dissertation entwickelten Modelle. Diese wurden dem Gesprächspartner einzeln vorgestellt. Dabei wurde auch der Erfüllungsgrad der in Kapitel 4.1 definierten formal-konzeptionellen und inhaltlichen Anforderungen der Modelle diskutiert. Die Zentis Fruchtwelt GmbH & Co setzt seit mehreren Jahren Low-Code-Plattformen ein und konnte somit aufgrund hoher Praxisexpertise wertvolle Beiträge zur Validierung des Regelwerks liefern. Durch das Interview war es möglich, die Ergebnisse der

Dissertation aus einer Praxisperspektive kritisch zu hinterfragen und den angestrebten Nutzen für die Anwendung im Unternehmen zu validieren.

9.2.1 Beschreibung des Anwendungsfeldes

Zentis, gegründet im Jahr 1893, ist ein technologieorientiertes, global agierendes Familienunternehmen, das sich auf die Verarbeitung und Veredelung von Früchten sowie anderen natürlichen Rohstoffen spezialisiert hat. Das Unternehmen bietet ein breites Produktportfolio, das auf natürlichen Inhaltsstoffen basiert und eine Vielzahl von Kreationen umfasst. Mit sechs Produktionsstandorten in Deutschland, Polen, Ungarn und den USA und einem Team von über 2.000 Mitarbeitenden ist Zentis in über 50 Ländern weltweit aktiv. Das Unternehmen arbeitet eng mit seinen Partnern zusammen, um maßgeschneiderte Lösungen für Start-ups, lokale Kunden und internationale Konzerne zu entwickeln. [1]

Zentis setzt zur Anwendungsentwicklung auf die Low-Code-Plattform Microsoft Power Plattform. Schwerpunktmäßig kommen dabei Power Apps, Power Automate und Power BI zum Einsatz. Zusätzlich wird UIPath für Robotic Process Automation (RPA) verwendet. Der Einsatz dieser Low-Code-Plattformen soll die Effizienz und Flexibilität bei der Entwicklung von Anwendungen und Automatisierungslösungen steigern.

Die Nutzung von RPA über UIPath erfolgt zentral durch die IT-Abteilung. Obwohl RPA viele Vorteile bietet, sind die Prozesse oft komplex und erfordern besondere Sicherheitsmaßnahmen. Daher wurde entschieden, die Entwicklung von RPA zentralisiert bei der IT zu belassen, während die Microsoft Power Plattform dezentral eingesetzt wird.

Zu Beginn der Einführung der Microsoft Power Plattform hat die IT-Abteilung die Plattform kennengelernt und erste Lösungen entwickelt sowie implementiert. Anschließend wurden Schulungen durchgeführt, um Mitarbeitende aus verschiedenen Fachbereichen – die sogenannten Business Champions – in die Lage zu versetzen, eigenständig Lösungen zu erstellen und anzupassen. Im Produktionsbereich wurden bereits über 200 Anwendungen implementiert. Die entstandene Community wächst stetig, und obwohl nicht täglich neue Lösungen erstellt werden, zeigt sich eine kontinuierliche Weiterentwicklung. Dieses Vorgehen ermöglichte einen sanften Einstieg in das Thema und fördert die Selbstständigkeit der Mitarbeitenden.

Das Ziel von Zentis ist es, den Mitarbeitenden die notwendigen Fähigkeiten und das richtige Mindset zu vermitteln, um eigenständig mit den neuen Tools arbeiten zu können. Dies erfordert auch ein Umdenken bei den Vorgesetzten, die den Mitarbeitenden Zeit und Ressourcen zur Verfügung stellen müssen, damit diese sich in die neuen Technologien einarbeiten und dadurch langfristig effizienter arbeiten können. Durch

[1] Informationen wurden der Website des Unternehmens entnommen: https://www.zentis.de/de/industrie/ueber-uns (Link zuletzt geprüft: 30.05.2024).

gezielte Veranstaltungen und Schulungen wurde das Bewusstsein auf höheren Führungsebenen geschaffen, dass die Digitalisierung und der Einsatz von Low-Code-Plattformen entscheidend sind, um die wachsende Flut an Anforderungen an die Fachbereiche optimal zu bewältigen.

Seit zwei Monaten nutzt Zentis auch Microsoft Copilot, was den Einsatz von Low-Code noch weiter erleichtern soll. Diese Unterstützung durch KI soll es noch einfacher machen, Anwendungen zu entwickeln und Prozesse zu automatisieren.

9.2.2 Darstellung der Ergebnisse

Im Rahmen der Ergebnispräsentation wurden die Modelle zunächst einzeln vorgestellt. Darauf aufbauend konnte die Anwendbarkeit in der Praxis mit dem Experten intensiv diskutiert werden. Nachstehend werden die Erkenntnisse dieser Diskussion entlang der Modelle dargelegt. Das Kapitel schließt mit der Bewertung des ganzheitlichen Regelwerks durch den Interviewpartner.

In einem ersten Schritt wurde der in Kapitel 5 entwickelte morphologische Kasten zur Beschreibung verschiedener Low-Code-Anwendungsfalltypen vorgestellt. Dabei erläuterte die Autorin die einzelnen Merkmale sowie deren spezifische Ausprägungen. Da Zentis schon vielseitige Erfahrungen mit Low-Code-Plattformen – vor allem bei Typ I und Typ II – sammeln konnte, stand eine allgemeine Bewertung des Modells im Mittelpunkt der Diskussion.

Im ersten Schritt wurde das Merkmal „Geschäftskritikalität“ besprochen. Der Experte konnte die verschiedenen Low-Code-Anwendungsfälle den unterschiedlichen Ausprägungen zuordnen. Ein Beispiel für eine unkritische Anwendung bei Zentis ist das Stellen eines Bewirtungsantrags, wenn ein Besprechungsraum mit Getränken ausgestattet werden soll. Eine kritische Anwendung wurde beispielsweise entwickelt, um bestimmte Vorschriften bei der Anlagenprüfung in der Produktion einzuhalten. Derzeit sind keine geschäftskritischen Anwendungen im Einsatz.

Zum Merkmal der „Nutzung“ bestätigte der Experte, dass es Anwendungen gibt, die nur teamintern genutzt werden, aber auch Anwendungen, die Abläufe betreffen, an denen fünf oder mehr Abteilungen beteiligt sind. Standortübergreifende Anwendungen seien bisher nicht entwickelt worden.

Das Merkmal „Lebensdauer“ wurde vom Experten direkt mit dem Merkmal „Geschäftskritikalität“ verknüpft. Er bestätigte, dass kritische Prozesse sich seltener ändern als unkritische und beurteilte die Einteilung in „wenige Jahre“, „mehrere Jahre“ und „viele Jahre“ als sinnvoll.

In Bezug auf die Schnittstellen betonte der Experte, dass eine der Herausforderungen bei Low-Code darin bestehe, die richtigen Daten sinnvoll zu integrieren. Deshalb werde bei Zentis derzeit eine „Azure Data Factory“ aufgebaut. Dabei würden Daten aus verschiedenen Quellen über Datenpipelines gesammelt und in einem zentralen Datentopf zusammengeführt. Dieser zentrale Datentopf solle helfen, Daten zielgerichteter mit standardisierten Schnittstellen anbinden zu können, sodass in Zukunft

weniger konfigurierbare oder individuelle Schnittstellen notwendig sind. Der Experte bewertete die Ausprägungen „standardisiert“, „konfigurierbar“ und „individuell“ als sinnvoll und bestätigte, dass mit allen drei Schnittstellenarten gearbeitet wurde.

Da die Anwendungen bei Zentis mit der Microsoft Power Plattform umgesetzt werden, ist die Anpassungsfähigkeit auf „Low-Code“ begrenzt, da der „Source Code“ nicht angepasst werden kann. Der Experte beurteilte die Anpassungsfähigkeit im „Source Code“ jedoch als sinnvolle Möglichkeit, den Funktionsumfang zu erweitern und standortübergreifende Anwendungsfälle mit Low-Code umzusetzen.

In Bezug auf die Programmierfähigkeiten stimmte der Experte zu, dass die unterschiedlichen Low-Code-Anwendungsfälle in ihrer Komplexität stark variieren und daher „keine“, „wenige“ oder „erfahrene“ Programmierfähigkeiten erforderlich sind. Er betonte jedoch, dass auch weniger komplexe Anwendungen oft von der IT-Abteilung mit erfahrenen Programmierfähigkeiten entwickelt worden seien, weil sich kein Fachbereich dafür verantwortlich gefühlt habe. Bei der Implementierung von Low-Code-Anwendungen gehe es nicht nur um die Komplexität, sondern auch um die verfügbaren Ressourcen.

Zusammenfassend betonte der Experte, dass die ausgewählten Merkmale geeignet seien, um ein Verständnis für die unterschiedlichen Typen zu entwickeln. Dabei hob er hervor, dass die Unterscheidung anhand der Merkmale nicht immer klar trennscharf sei, da beispielsweise auch unkritische Anwendungsfälle standortübergreifend existieren können und erfahrene Programmierfähigkeiten erfordern würden. Die Typenbildung hilft dabei, Anwendungsfälle zu strukturieren, lässt jedoch Raum für Anpassungen und spezifische Anforderungen, die außerhalb der Norm liegen.

Abschließend konnte der Experte die reale Existenz der drei Typen „Entwicklung durch Fachbereiche“, „Entwicklung durch Fachbereiche und IT-Spezialist*innen“ sowie „Entwicklung durch IT-Spezialist*innen“ bestätigen. Im gemeinsamen Diskurs wurde festgehalten, dass die Entwicklung durch die jeweiligen Bereiche nur stattfinden kann, wenn genügend Kapazitäten dafür vorgesehen sind und diese an die Low-Code-Entwicklung herangeführt werden. Dennoch wurde bestätigt, dass die Typenbildung aus wissenschaftlicher Perspektive notwendig ist, um die Auswirkungen auf die einzuhaltenden Regeln anschaulich und praxisnah erklären zu können und geeignete Maßnahmen ableiten zu können.

In einem zweiten Schritt wurde das Low-Code-Regelwerk (siehe Kapitel 6) vorgestellt. Der Experte hob die grafische Darstellung dieses Modells positiv hervor und betonte, dass alle Aspekte eine hohe Relevanz für die Implementierung und den Betrieb von Low-Code-Anwendungen aufweisen. Die Diskussion zeigte, dass in der Praxis alle aufgezeigten Regeln bekannt sind, jedoch bisher keine grafische Strukturierung dieser Aspekte verfügbar war. Diese Feststellung bestätigt die im Rahmen des Dissertationsvorhabens aufgedeckte Forschungslücke. Zusammenfassend bestätigte der Validierungspartner die praktische Relevanz aller identifizierten Regeln. Im Verlauf der Diskussion wurden einzelne Regeln genauer besprochen. Wie im vorangegangenen Fallbeispiel beschrieben, spielten vor allem die Regel zu „Ressourcen“ und

„Anspruchsgruppen“ eine wichtige Rolle, da ausreichend Kapazitäten notwendig sind und die Mitarbeitenden im Bereich Low-Code geschult werden müssen. Zudem sollte der Austausch zwischen den unterschiedlichen Anspruchsgruppen gezielt gefördert werden, was bei Zentis durch den Aufbau der „Business Champions“ erfolgt. Das Gespräch machte zudem deutlich, dass ein geregelter Support-Prozess für die erfolgreiche Umsetzung bei Zentis von großer Bedeutung war.

Im dritten Schritt wurde die Vorgehensweise zur Herleitung der Wirkbeziehungen zwischen den Low-Code-Anwendungsfalltypen und dem Regelwerk dargestellt (siehe Kapitel 7). Der Experte beurteilte den Prozess zur Datenerhebung als sehr praxisnah, wodurch die Herleitung des Modells dem Anspruch einer explorativen Forschungskonzeption gerecht wird. Im Rahmen des Gesprächs wurde darauf verzichtet, jede Wirkungsbeziehung im Detail zu diskutieren. Dennoch wurden generelle Stimmungsbilder ausgetauscht und bestätigt, dass eine homogene Einschätzung in der Industrie vorliegt.

Im vierten Schritt wurden die Gestaltungsmaßnahmen und das Vorgehen zur Anwendung des Regelwerks (siehe Kapitel 8) präsentiert. Hierbei wurde dem Experten analog zum Erklärungsmodell zunächst die Vorgehensweise zur Herleitung erläutert. Positiv hervorgehoben wurde die Identifikation geeigneter Gestaltungsmaßnahmen in Anlehnung an die Expert*inneninterviews. Eine detaillierte Diskussion aller 60 Gestaltungsmaßnahmen wurde aus Zeitgründen nicht angestrebt. Die Gültigkeit und Praxistauglichkeit dieser Maßnahmen konnten jedoch mithilfe der vorangegangenen Expert*inneninterviews bestätigt werden. Bezogen auf die Anwendung des Regelwerks betonte der Experte, dass es wichtig sei zu klären, in welcher Phase sich ein Unternehmen befindet, das Low-Code einführen möchte. Für Unternehmen, die bereits in der Industrialisierungsphase sind, bietet die Überprüfung und Feinjustierung der bestehenden Prozesse einen großen Nutzen. Zentis befindet sich als Unternehmen am Anfang der Industrialisierungsphase.

Da bei Zentis die Anwendungsfalltypen I und II vorkommen, wird der Experte die für diese Typen spezifischen Gestaltungsmaßnahmen im Detail durchgehen und innerhalb des Unternehmens überprüfen, ob alle relevanten Punkte bereits berücksichtigt wurden. Um die praktische Anwendbarkeit der Gestaltungsmaßnahmen zu verdeutlichen, können konkrete Beispiele aus der bisherigen Umsetzung von Low-Code-Anwendungen herangezogen werden.

Bei Anwendungsfalltyp I handelt es sich um die Entwicklung von Low-Code-Anwendungen durch die Fachbereiche. Ein Beispiel hierfür ist der Aufbau einer zentralen Dateninfrastruktur mithilfe der Azure Data Factory. Die IT stellt dafür standardisierte Schnittstellen bereit (Maßnahme O4 siehe A.14), die den Fachbereichen einen eigenständigen Zugriff ermöglichen. Mithilfe gezielter Schulungen wurden die Fachbereiche befähigt, BI-Reports und Dashboards selbstständig zu erstellen und individuell anzupassen (Maßnahme O1 siehe A.14).

Bei Anwendungsfalltyp II erfolgt die Entwicklung von Low-Code-Anwendungen durch eine enge Zusammenarbeit zwischen IT und Fachbereichen. Ein konkretes Beispiel ist

das digitale Checklisten-Management in der Produktion. Diese Anwendung wurde zunächst gemeinsam von IT und Fachbereichen entwickelt, um papierbasierte Prüfprozesse zu digitalisieren. Mitarbeitende können heute Prüfungen direkt über eine App dokumentieren, indem sie Barcodes scannen und Ergebnisse in Echtzeit erfassen. Nach der Inbetriebnahme durch die IT (Maßnahme O30 siehe A.14) wurden die Fachbereiche geschult (Maßnahme O1 siehe A.14), um kleinere Anpassungen wie das Hinzufügen neuer Prüfpunkte eigenständig vorzunehmen. Die Verantwortung der Anwendung wurde somit an den Fachbereich übergeben (Maßnahme O19 siehe A.14). Diese Zusammenarbeit gewährleistet, dass die Anwendungen sowohl technisch robust als auch fachlich relevant sind, während die Fachbereiche langfristig in der Lage sind, die Lösung eigenständig weiterzuentwickeln.

Ein zentraler Erfolgsfaktor bei beiden Anwendungsfalltypen ist der gezielte Austausch zwischen den Entwickler*innen der Fachbereiche, den sogenannten Business Champions. Dieser wird bei Zentis durch regelmäßige Meetings gezielt gefördert (Maßnahme O25 siehe A.14). Innerhalb dieser Community werden Best Practices geteilt, Herausforderungen gemeinsam gelöst und ein nachhaltiger Wissensaufbau unterstützt, der die kontinuierliche Weiterentwicklung der Low-Code-Kompetenzen sicherstellt (Maßnahme O2 siehe A.14).

Die genannten Beispiele verdeutlichen, wie sich die Umsetzbarkeit der in der Arbeit entwickelten Gestaltungsmaßnahmen in der Praxis zeigt und wie die Zusammenarbeit zwischen IT-Abteilung und Fachbereichen erfolgreich gestaltet werden kann. Die strukturierte Aufbereitung der Low-Code-Regeln und anwendungsfallspezifischen Gestaltungsmaßnahmen liefert Zentis ein praxiserprobtes Vorgehen, um die bereits umgesetzten Maßnahmen zu erweitern und die bestehende IT-Governance weiterzuentwickeln. Damit bieten das Regelwerk und die spezifischen Gestaltungsmaßnahmen einen praktischen und nachhaltigen Mehrwert.

9.2.3 Bewertung des Fallbeispiels

Den Abschluss des Validierungsinterviews bildete die schrittweise Bewertung der Teilmodelle durch die in Kapitel 4.1 definierten formal-konzeptionellen und inhaltlichen Anforderungen an das Regelwerk (siehe Abbildung 9-2).

Formale Anforderungen		
Validität	Sind die Modelle interdisziplinär verständlich und auf ähnliche Probleme übertragbar?	✓
Reliabilität	Führen die Modelle bei wiederholter Anwendung unter gleichen Bedingungen zu gleichen Ergebnissen?	✓
Utilität	Sind die Modelle nützlich, verständlich und für die praktische Anwendbarkeit definiert?	✓
Inhaltliche Anforderungen		
Merkmale und Ausprägungen	Sind die Merkmale und Ausprägungen vollständig, praxistauglich und überschneidungsfrei?	✓
Heterogenität der Typen	Können die identifizierten Typen deutlich voneinander abgegrenzt werden?	✓
Reale Existenz der Typen	Exisiteren die Typen in der Praxis und sind die Merkmalskombinationen logisch kohärent, empirisch überprüfbar und praktisch anwendbar?	✓
Regelwerk	Berücksichtigt das Regelwerk alle relevanten und tatsächlich existenten Risiken von Low-Code?	✓
Wirkbeziehungen	Wurde die wechselseitige Wirkbeziehung der Low-Code Anwendungsfalltypen und Regeln verständlich erklärt?	✓
Gestaltungs-maßnahmen	Liefert das Regelwerk sowie die anwendungsfallspezifischen Gestaltungsmaßnahmen einen Mehrwert in der Praxis?	✓
Innere Konsistenz	Stellt die Gesamtheit der Teilmodelle ein kohärentes Bild der komplexen Realität dar?	✓
Visualisierung	Trägt die Visualisierung dazu bei, die Modelle klar und übersichtlich zu vermitteln?	✓
Anpassungs-fähigkeit	Sind die Modelle so gestaltet, dass sie vor dem Hintergrund eines technologischen Wandels anpassungsfähig sind?	✓

Abbildung 9-2: Bewertung der formal-konzeptionellen und inhaltlichen Anforderungen durch die Zentis GmbH & Co. KG (eigene Darstellung)

Im Zuge der Bewertung formaler Anforderungen konnte deren Validität bereits in den vorangegangenen Ausführungen nachgewiesen werden. Auch die Reliabilität der Modelle wurden vom Experten als gegeben bewertet. Basierend auf dem Vorgehen zur Anwendung des Regelwerks wurde betont, dass die Modelle verständlich und in der Praxis anwendbar sind. Die Erfüllung der formalen Anforderungen konnte durch das Validierungsgespräch festgestellt werden.

Eine Einschätzung der Vielzahl inhaltlicher Anforderungen wurde bereits im vorangegangenen Kapitel herausgestellt. Demnach sind die identifizierten Merkmale und Merkmalsausprägungen sowie die Dimensionen und Regeln der beiden Beschreibungsmodelle kontextuell vollständig und für die adressierte Zielgruppe praxistauglich. Auch die reale Existenz und die Heterogenität der Typen wurden im Rahmen des Interviews hinreichend bestätigt. Darüber hinaus ist auch die Darstellung der Wirkintensitäten verständlich erklärt, um ein Verständnis der notwendigen Maßnahmen zur Einhaltung der Regeln je Anwendungsfalltyp zu vermitteln. Durch die Erfüllung der inhaltlichen Anforderungen der einzelnen Modelle weist das Low-Code-Regelwerk einen hohen Nutzen für die Praxis auf. Ebenfalls positiv hervorgehoben wurde die Visualisierung der Modelle, um Sachverhalte unmissverständlich zu transportieren. Abschließend konnten zudem die Fragen nach innerer Konsistenz und Adaptionsfähigkeit der Modelle vor dem Hintergrund eines kontinuierlichen Wandels bejaht werden.

Die Ergebnisse dieser Dissertation wurden anhand des Fallbeispiels der Zentis GmbH & Co. KG intensiv im praktischen Anwendungszusammenhang diskutiert. Dabei lieferten die umfassenden Erfahrungen des Experten im Low-Code-Bereich wertvolle Einblicke. Diese Einblicke waren entscheidend, um den Praxisnutzen und die Anwendbarkeit des Regelwerks zu validieren.

9.3 Fallbeispiel: Mittelständischer Automobilzulieferer

Das zweite Interview zur Validierung der Ergebnisse wurde mit dem Director IT International eines mittelständischen Automobilzulieferers geführt. Im ersten Teil des Interviews beschrieb der Ansprechpartner den aktuellen Status der Low-Code-Aktivitäten innerhalb des Unternehmens. Anschließend wurden die Inhalte der Dissertationsschrift sowie die einzelnen Modelle im Detail vorgestellt. Zum Abschluss wurden die formal-konzeptionellen und inhaltlichen Anforderungen der partialen Modelle erläutert und diskutiert.

Innerhalb des Unternehmens wurden bereits dezentrale Automatisierungsprojekte mit RPA ausgerollt. Die RPA-Projekte werden von der IT umgesetzt, indem sie hauptsächlich lokal organisiert und zentral koordiniert werden. Durch den Einsatz von RPA haben einige Standorte bereits erhebliche Einsparungen durch die Automatisierung von Prozessen erzielt. Zukünftig ist die Implementierung einer Low-Code-Plattform geplant.

Da das zuvor untersuchte Fallbeispiel ein Unternehmen in der Industrialisierungsphase war, ist das Fallbeispiel des Automobilzulieferers – ein Unternehmen in der Initialisierungsphase – ein passender Anwendungsbereich für das Dissertationsvorhaben. Der Gesprächspartner lieferte wertvolle Beiträge zur Validierung des Regelwerks. Das Interview ermöglichte es, die Ergebnisse des Dissertationsvorhabens aus einer praxisnahen Perspektive zu betrachten und den angestrebten Nutzen für die Anwendung im Unternehmen zu bestätigen.

9.3.1 Beschreibung des Anwendungsfeldes

Der Automobilzulieferer ist eine weltweit tätige Unternehmensgruppe mit Niederlassungen in Europa, Nordamerika und Asien. An 21 Standorten mit über 6.500 Mitarbeitenden entwickelt, produziert und vertreibt das Unternehmen innovative Systeme und Komponenten für die Automobilindustrie. In über 100 Jahren Unternehmensgeschichte hat es sich eine globale Präsenz sowie die technologische und marktführende Stellung in wichtigen Produktbereichen des Fahrzeugs erarbeitet.

Das Unternehmen befindet sich derzeit in einer Phase der Weiterentwicklung und Optimierung seiner Automatisierungsstrategien. Teil der Automatisierungsstrategien sind die Entwicklung mit RPA und Low-Code. Während RPA schon aktiv umgesetzt wird und wächst, befindet sich die Entwicklung mit Low-Code noch im Aufbau.

Die RPA-Aktivitäten werden an den einzelnen Standorten organisiert und zentral von der internationalen IT koordiniert. Die technische Umsetzung von RPA erfolgt durch die IT-Abteilung in enger Zusammenarbeit mit den Fachbereichen.

Derzeit wird die Nutzung von Low-Code-Plattformen vorbereitet. Zunächst konzentriert sich das Unternehmen darauf, die IT-Architektur durch größere IT-Systeme zu modernisieren. Danach sollen die verbleibenden Funktionalitätslücken, die nicht durch diese Systeme abgedeckt werden, durch selbstentwickelte Low-Code-Anwendungen geschlossen werden. Das Unternehmen befindet sich in der Initialisierungsphase von Low-Code. Zu dieser Phase gehören Workshops und Schulungen, die von der Autorin gemeinsam mit dem interviewten Experten durchgeführt wurden. Diese sollen das Bewusstsein für Automatisierung und Low-Code-Lösungen schärfen und den direkten Nutzen für die Fachbereiche aufzeigen.

Es wurden bereits Abschlussarbeiten und erste Implementierungsversuche durchgeführt sowie zahlreiche Anwendungsfälle identifiziert. Nach der Modernisierung der IT-Architektur plant das Unternehmen, die Low-Code-Aktivitäten weiter auszubauen.

9.3.2 Darstellung der Ergebnisse

Nachdem der Interviewpartner den Anwendungsfall dargelegt hatte, wurde der Schwerpunkt der Dissertation vorgestellt. Dabei wurden die identifizierte Problemstellung und die entwickelten Modelle der Arbeit erörtert und diskutiert. Die Ergebnisse der Validierung werden im Folgenden beschrieben.

Zunächst wurden der in Kapitel 5 entwickelte morphologische Kasten und die Typisierung zur Beschreibung verschiedener Low-Code-Anwendungsfalltypen vorgestellt. Der morphologische Kasten und die anschließende Typisierung fanden bereits Anwendung, da sie von der Autorin in einem Workshop innerhalb des Unternehmens präsentiert wurde. Das Modell wurde genutzt, um einen ersten Überblick über mögliche Low-Code-Anwendungsfälle zu erhalten und diese zu strukturieren. Im Anschluss erstellte das Unternehmen eine Liste von Anwendungsfällen mit kurzen Beschreibungen.

Der Experte betonte, dass nicht alle Anwendungsfälle strikt den Ausprägungen eines Typs zugeordnet werden können. Es gebe beispielsweise geschäftskritische Anwendungsfälle, die nur in einer Abteilung verwendet werden und für einen Zeitraum von drei bis fünf Jahren im Einsatz sind, ohne dass Programmierfähigkeiten erforderlich sind. Jeder Anwendungsfall könne in gewissem Maße von den üblichen Ausprägungen abweichen. Selten gebe es jedoch Anwendungsfälle, die komplett aus dem Rahmen fallen. Trotz kleiner Abweichungen lasse sich jeder Anwendungsfall einem Typ zuordnen. Die Typisierung biete daher einen ausreichend praxistauglichen Rahmen, um alle im Unternehmen anfallenden Low-Code-Anwendungsfälle voneinander abzugrenzen.

Im nächsten Schritt wurde das Low-Code-Regelwerk vorgestellt (siehe Kapitel 6). Der Experte schätzte das Regelwerk als vollständig ein, betonte jedoch, dass man jede Regel noch weiter vertiefen könnte, was die Notwendigkeit der Gestaltungsmaßnahmen (siehe Kapitel 8.1) verdeutlicht. Besonders hervorgehoben wurden die Regeln für Ressourcen, Qualität, Lösungserstellung und Betrieb.

In Bezug auf die Ressourcen betonte der Experte, dass es wichtig sei, die Fachbereiche durch Schulungen zu unterstützen und durch ein Rollenmodell festzulegen, wer welche Befugnisse hat. Die Bedeutung der Qualitätskontrolle wurde ebenfalls betont, da es bei bestimmten Anwendungen sinnvoll sei, wenn die IT die Entwicklung überprüft. Bei der Lösungserstellung wurde reflektiert, dass ein automatisiertes Testing die optimale Umsetzung wäre, dies jedoch teilweise zu aufwendig sei, um es für alle Low-Code-Anwendungen vollumfänglich umzusetzen.

Eine weitere Problematik, die bei dem Unternehmen zu Beginn der RPA-Implementierung nicht ausreichend berücksichtigt wurde, betrifft den Betrieb und das Versionsmanagement der Anwendungen. Wenn ein Fachbereich eine Anwendung erstellt, möglicherweise ohne die IT-Abteilung zu konsultieren, können Probleme auftreten. Die IT-Abteilung installiert regelmäßig Updates oder nimmt andere Änderungen an der Software vor, was dazu führen kann, dass die RPA-Anwendung nicht mehr funktioniert. Ein typisches Beispiel ist ein Windows-Update, das von Microsoft bereitgestellt wird und nicht vermieden werden kann. Der Experte bestätigte, dass das Low-Code-Regelwerk sicherstellt, dass Themen, wie das Versionsmanagement, von Beginn an mitberücksichtigt werden.

Nach der Diskussion des Low-Code-Regelwerks wurde die Vorgehensweise zur Herleitung der einzelnen Wirkungsbeziehungen erläutert (siehe Kapitel 7). Der Gesprächspartner lobte die praxisnahe Durchführung der Interviews. Der Experte wies dabei auf kleinere unternehmensspezifische Abweichungen hin, beispielsweise bei der Wirkbeziehung zwischen IT-Architektur und standortübergreifenden Anwendungen. Der Experte betonte, dass es bei großen und diversifizierten Unternehmen wie Bosch oder Siemens viele unterschiedliche Geschäftsfälle und Standorte gebe, die jeweils unterschiedliche IT-Architektur hätten. Da der Automobilzulieferer nicht so diversifiziert sei und fast überall die gleichen Produkte herstelle, verfüge das Unternehmen im Vergleich zu anderen internationalen Unternehmen über eine sehr einheitliche IT-Architektur. Durch dieses Beispiel verdeutlichte der Experte, dass jedes Unternehmen die Wirkbeziehungen individuell prüfen sollte. Insgesamt wurde jedoch besonders die Aufteilung in typenunabhängige und typenspezifische Maßnahmen als sehr sinnvoll angesehen, da dadurch eine deutlich zielgerichtetere Implementierung ermöglicht werde. Diese Aufteilung hilft Unternehmen, zu identifizieren, welche Maßnahmen tatsächlich notwendig sind und wo Aufwände eingespart werden können.

Im weiteren Verlauf des Interviews wurden die Gestaltungsmaßnahmen vorgestellt (siehe Kapitel 8). Diese wurden als äußerst nützlich für Unternehmen bewertet. Besonders hervorgehoben wurden die Gestaltungsmaßnahmen für Typ I und Typ II, da die reduzierte Anzahl an Maßnahmen den Unternehmen hilft, die Implementierung von Low-Code-Anwendungen gezielt und effizient anzugehen. Der Experte merkte an, dass die umfassende Liste der Gestaltungsmaßnahmen für Typ III den üblichen Regeln der Softwareentwicklung innerhalb der IT-Abteilung entspreche. Die Maßnahmen für Typ III sind besonders relevant für Unternehmen, deren IT-Abteilungen bisher wenig Erfahrung mit Softwareentwicklung haben.

Insgesamt bewertete der Experte den Nutzen und die praktische Anwendbarkeit des Regelwerks als sehr hoch. Besonders die präzisen Gestaltungsmaßnahmen, die aus praktischen Erfahrungen und Herausforderungen anderer Low-Code-Anwender*innen resultieren, wurden als äußerst wertvoll hervorgehoben. Da die Ergebnisse modular aufgebaut und individuell an das Unternehmen und den technischen Fortschritt angepasst werden können, wurde die Aufbereitung der Ergebnisse als eine große Unterstützung im Einführungsprozess angesehen.

Analog zum Vorgehen zur Anwendung des Regelwerks hat der Automobilzulieferer im ersten Schritt bereits mögliche Low-Code-Anwendungsfälle gesammelt und den drei vorgeschlagenen Typen zugeteilt. Zwei konkrete Beispiele aus der eigenen Praxis zeigen, wie die Ergebnisse dieser Arbeit bei den bereits implementierten RPA-Anwendungen innerhalb des Unternehmens genutzt werden können.

Ein RPA-Bot wurde eingesetzt, um Daten aus einem Microsoft System zu extrahieren. Nach einem Windows-Versionsupdate durch Microsoft funktionierte der Bot nicht mehr, da sich die Benutzeroberfläche des IT-Systems verändert hatte. Dieses Problem hätte durch automatisiertes Testing (Maßnahme T9 siehe A.14) und die Überprüfung der Schnittstellenkonfiguration bei Versionsupdates (Maßnahme O40 siehe A.14) verhindert werden können. Mit einer Integration solcher Maßnahmen in die IT-Governance wäre die Funktionsfähigkeit des Bots trotz des Updates sichergestellt gewesen.

Ein weiteres Beispiel betrifft einen RPA-Bot, der Daten aus einem IT-System extrahierte und weiterverarbeitete. Als die IT die Zugriffsrechte des IT-Systems anpasste, konnte der Bot nicht mehr auf die benötigten Daten zugreifen. Dieses Problem hätte durch die Definition anwendungsspezifischer Regeln zum Datenumgang und -zugriff (Maßnahme O43 siehe A.14) vermieden werden können, indem die Abhängigkeiten des Bots von des IT-Systems dokumentiert und berücksichtigt worden wären.

Die genannten RPA-Beispiele aus der Praxis zeigen, wie durch die Umsetzung solcher Maßnahmen die Zusammenarbeit zwischen Fachbereichen und IT verbessert und die Zuverlässigkeit der RPA-Anwendungen nachhaltig gesteigert werden kann. Im nächsten Schritt plant der Automobilzulieferer, die typenunabhängigen sowie typenspezifischen Gestaltungsmaßnahmen für jeden Low-Code-Anwendungsfalltyp zu identifizieren. Basierend darauf werden die Gestaltungsmaßnahmen für die drei unterschiedlichen Anwendungsfalltypen etabliert und in die bestehende IT-Governance integriert.

9.3.3 Bewertung des Fallbeispiels

Der Abschluss des Validierungsinterviews bestand in der schrittweisen Bewertung der in Kapitel 4.1 definierten formal-konzeptionellen und inhaltlichen Anforderungen an das Regelwerk sowie dessen Modelle (siehe Abbildung 9-3). Alle formalen Anforderungen wurden vom Experten als erfüllt angesehen, wobei besonders die Struktur der einzelnen Modelle und die präzisen Gestaltungsmaßnahmen hervorgehoben wurden.

Formale Anforderungen		
Validität	Sind die Modelle interdisziplinär verständlich und auf ähnliche Probleme übertragbar?	✓
Reliabilität	Führen die Modelle bei wiederholter Anwendung unter gleichen Bedingungen zu gleichen Ergebnissen?	✓
Utilität	Sind die Modelle nützlich, verständlich und für die praktische Anwendbarkeit definiert?	✓
Inhaltliche Anforderungen		
Merkmale und Ausprägungen	Sind die Merkmale und Ausprägungen vollständig, praxistauglich und überschneidungsfrei?	✓
Heterogenität der Typen	Können die identifizierten Typen deutlich voneinander abgegrenzt werden?	✓
Reale Existenz der Typen	Exisiteren die Typen in der Praxis und sind die Merkmalskombinationen logisch kohärent, empirisch überprüfbar und praktisch anwendbar?	✓
Regelwerk	Berücksichtigt das Regelwerk alle relevanten und tatsächlich existenten Risiken von Low-Code?	✓
Wirkbeziehungen	Wurde die wechselseitige Wirkbeziehung der Low-Code Anwendungsfalltypen und Regeln verständlich erklärt?	✓
Gestaltungs-maßnahmen	Liefert das Regelwerk sowie die anwendungsfallspezifischen Gestaltungsmaßnahmen einen Mehrwert in der Praxis?	✓
Innere Konsistenz	Stellt die Gesamtheit der Teilmodelle ein kohärentes Bild der komplexen Realität dar?	✓
Visualisierung	Trägt die Visualisierung dazu bei, die Modelle klar und übersichtlich zu vermitteln?	✓
Anpassungs-fähigkeit	Sind die Modelle so gestaltet, dass sie vor dem Hintergrund eines technologischen Wandels anpassungsfähig sind?	✓

Abbildung 9-3: Bewertung der formal-konzeptionellen und inhaltlichen Anforderungen durch das Unternehmen (eigene Darstellung)

Bei der inhaltlichen Bewertung der Typisierung wurden sowohl die Praxistauglichkeit als auch die Vollständigkeit der Merkmale und Merkmalsausprägungen bestätigt. Auch wenn nicht jeder Low-Code-Anwendungsfall allen Ausprägungen der Typisierung entspricht, lässt sich jeder Anwendungsfall einem Typ zuordnen. Der Interviewpartner betonte zudem, dass die einzelnen Typen mit allen Ausprägungen eine reale Existenz haben.

Auch das Low-Code-Regelwerk wurde als vollständig und praxistauglich bewertet. Besonders positiv wurde die Struktur des Regelwerks hervorgehoben. Der Experte bestätigte, dass das Low-Code-Regelwerk sicherstellt, dass alle Herausforderungen und Risiken von Low-Code von Beginn an berücksichtigt werden.

Bei der Validierung der Wirkbeziehungen wurde die Unterscheidung in typenunabhängige und typenspezifische Maßnahmen als sinnvoll erachtet, und der starke Praxisbezug aufgrund der Expert*inneninterviews wurde gelobt. Trotz möglicher Ausreißer bei der Typisierung oder den Wirkbeziehungen können durch die Betrachtung der Wirkbeziehungen die notwendigen Maßnahmen identifiziert werden. Abschließend wurde festgehalten, dass die Gestaltungsmaßnahmen sowie die Anwendung des gesamten Regelwerks im Implementierungsprozess einen hohen praktischen Nutzen haben und angesichts des technologischen Fortschritts anpassungsfähig gestaltet sind.

Anhand des Fallbeispiels wurden die Ergebnisse der vorliegenden Dissertationsschrift intensiv aus der Anwenderperspektive diskutiert. Zusammenfassend lässt sich festhalten, dass der Experte aufgrund seiner langjährigen Erfahrung und seiner kritischen Auseinandersetzung mit der Low-Code-Implementierung im eigenen Unternehmen maßgeblich dazu beitragen konnte, die inhaltliche Konsistenz der beiden Beschreibungsmodelle zu prüfen und den praktischen Nutzen des gesamten Regelwerks und der Gestaltungsmaßnahmen zu validieren.

10 Zusammenfassung und Ausblick

Im abschließenden Kapitel dieser Dissertation werden die Hauptergebnisse zusammengefasst, einer kritischen Betrachtung unterzogen und ihr Nutzen sowohl für die wissenschaftliche Gemeinschaft als auch die Praxis herausgestellt (siehe Kapitel 10.1). Weiterhin wird auf den weiteren Forschungsbedarf eingegangen, der sich aus den Erkenntnissen dieser Arbeit ergibt (siehe Kapitel 10.2).

10.1 Zusammenfassung

Die Motivation des vorliegenden Dissertationsvorhaben resultierte aus dem Praxisdefizit eines mangelnden Verständnisses von Low-Code-Anwendungsmöglichkeiten und einer fehlenden Governance beim Einsatz von Low-Code in Unternehmen. Insbesondere vor dem Hintergrund der wachsenden Nachfrage an digitalen Lösungen und dem gleichzeitigen Mangel an IT-Fachkräften bietet der Einsatz von Low-Code die Möglichkeit, Nicht-IT-Spezialist*innen digitale Lösungen entwickeln zu lassen, wodurch der Entwicklungsprozess beschleunigt und vereinfacht wird.

Ergebnisse aus der Forschung zeigen jedoch, dass trotz der Vorteile von Low-Code-Plattformen die Softwareentwicklung meist weiterhin bei den IT-Abteilungen liegt. Der Einsatz von Low-Code wird aufgrund von Datensicherheit- und Datenschutzbedenken gehemmt. Das fehlende Wissen über die technischen Möglichkeiten von Low-Code-Plattformen führt in Unternehmen zu einem fehlenden Verständnis über die Implementierung und den Betrieb von Low-Code-Anwendungen. Zusätzlich fehlt es organisatorisch an IT-Governance-Ansätzen, die den Low-Code Einsatz steuern und überwachen, was zu Sicherheitsrisiken und mangelnder Qualität der Anwendungen führt.

Dieses Dissertationsvorhaben zielte darauf ab, die Anwendung von Low-Code sowohl technisch als auch organisatorisch zu betrachten, um eine zielgerichtete Steuerung und Überwachung zu gewährleisten. Dazu wurde ein Regelwerk für Low-Code-Anwendungen in produzierenden Unternehmen entwickelt, das präzise Regeln und unterstützende Gestaltungsmaßnahmen für spezifische Low-Code-Anwendungsfälle umfasst und die bestehende IT-Governance innerhalb eines Unternehmens erweitern soll. Damit wird die zentrale Forschungsfrage beantwortet:

Welche Regeln sollte ein produzierendes Unternehmen für Low-Code-Anwendungen mindestens festlegen?

Die Entwicklung des Regelwerks mit anwendungsfallspezifischen Gestaltungsmaßnahmen basiert auf vier Teilzielen. Zunächst mussten die verschiedenen Low-Code-Anwendungsfälle typisiert und ein Low-Code-Regelwerk entwickelt werden. Anschließend wurden mithilfe von Wirkbeziehungen zwischen Anwendungsfalltyp und Low-Code-Regel, Gestaltungsmaßnahmen identifiziert, die Anwendungsfallspezifisch angewendet werden können. Im Folgenden wird eine Zusammenfassung der Teilziele und der erzielten Ergebnisse dieser Dissertation präsentiert.

Teilziel I: Typisierung von Low-Code-Anwendungsfällen

Die Motivation für das erste Modell entstand aus der Forschungslücke eines fehlenden Modells zur Beschreibung von Low-Code-Anwendungsfällen in produzierenden Unternehmen. Der Modellierungszweck orientierte sich hierbei an einem deskriptiven Erkenntnisziel (s. ZELEWSKI 2008, S. 24). Das Modellierungsziel bestand in der Identifikation von Merkmalen und Merkmalsausprägungen von Low-Code-Anwendungsfällen, um diese anhand von spezifischen Eigenschaften strukturiert beschreibbar und analysierbar zu machen. Anschließend wurden Verknüpfungen zwischen den Merkmalsausprägungen gebildet, welche die Ableitung konsistenter und real existierender Typen ermöglichten. Dadurch konnte die ersten Unterforschungsfrage beantwortet werden:

Wie können Low-Code-Anwendungsfälle in produzierenden Unternehmen beschrieben werden?

Das Ergebnis dieser Untersuchung mündete in der Erstellung eines morphologischen Kastens, der sechs Merkmale und siebzehn dazugehörige Merkmalsausprägungen umfasst. Die sorgfältige Verknüpfung dieser Ausprägungen ermöglichte es, logisch schlüssige Kombinationen zu bilden. Auf dieser Grundlage wurden drei Anwendungsfalltypen identifiziert, deren reale Existenz anschließend durch Praxisbeispiele in Form von Expert*inneninterviews bestätigt wurde. Dadurch wurde das identifizierte Praxisproblem adressiert: das bisher fehlende Verständnis verschiedener Low-Code-Anwendungsmöglichkeiten. Dieses erste Modell leistet somit einen entscheidenden Beitrag zum Verständnis des Einsatzes von Low-Code und bietet eine klare Orientierungshilfe für zukünftige Untersuchungen.

Teilziel II: Entwicklung von Low-Code-Regeln

Das zweite Teilziel der Dissertation konzentrierte sich auf die Entwicklung eines Low-Code-Regelwerks, das ebenfalls ein deskriptives Erkenntnisziel verfolgte. Dabei zielte das Modell darauf ab, die Risiken und Herausforderungen des Einsatzes von Low-Code zu identifizieren, um daraus Regeln abzuleiten, die den Risiken und Herausforderungen entgegenwirken. Die Unterforschungsfrage dieser Untersuchung lautete:

Wie können Regeln für Low-Code innerhalb eines produzierenden Unternehmens beschrieben werden?

Eine systematische Literaturrecherche ergab zunächst, dass spezifische Regeln und Governance-Ansätze für den Einsatz von Low-Code in produzierenden Unternehmen fehlen. Aus diesem Grund wurde die Literaturrecherche auf Risiken und Herausforderungen von Low-Code ausgeweitet. Durch die Anwendung der Inhaltsanalyse nach Mayring wurden Strukturierungsdimensionen abgeleitet und Low-Code-Regeln identifiziert und anschließend weiter spezifiziert. Das Ergebnis ist die Definition eines Low-Code-Regelwerks, das sich aus fünf Dimensionen und elf Regeln zusammensetzt. Durch dieses systematische Vorgehen wurde ein theoretisches Fundament geschaffen, das für die Integration der ersten beiden Beschreibungsmodelle von

entscheidender Bedeutung ist und somit einen umfassenden Rahmen für anwendungsfallspezifische Gestaltungsmaßnahmen bietet.

Teilziel III: Erklärung der wechselseitigen Wirkbeziehungen

Das dritte Teilziel der Dissertation fokussierte sich auf das Verständnis der wechselseitigen Wirkbeziehungen zwischen den zuvor entwickelten Beschreibungsmodellen zu Low-Code-Anwendungsfalltypen und Regeln. Das Modell zielt darauf ab, die realen Beziehungen zu erklären, in Übereinstimmung mit dem deskriptiven Ansatz von ZELEWSKI (s. ZELEWSKI 2008, S. 24). Die Modellierung strebte an, die Wirkbeziehungen zwischen den Low-Code-Anwendungsfalltypen und den Regeln zu identifizieren und deren Intensitäten stufenweise zu charakterisieren. Die Untersuchung ermöglichte die Beantwortung der dritten Unterforschungsfrage:

Wie können aus den Low-Code-Anwendungsfällen Wirkbeziehungen zu den Low-Code-Regeln erklärt werden?

Aufgrund begrenzter Literatur zu Low-Code-Anwendungsfällen und -Regeln zur Beantwortung dieser Frage wurde ein praxisorientierter Forschungsansatz gewählt. Durch qualitative Expert*inneninterviews wurden Erkenntnisse aus der Industrie gesammelt, um die Wirkbeziehungen zu identifizieren und deren Intensität zu charakterisieren. Dabei wurde untersucht, bei welchen Anwendungsfalltyp welche Maßnahme notwendig ist, um die zuvor definierten Regeln einzuhalten. Die Ergebnisse der Untersuchung offenbarten, dass es sowohl typenunabhängige Maßnahmen als auch typenspezifische Maßnahmen gibt. Diese Erkenntnisse ermöglichen es, fundierte Gestaltungsmaßnahmen abzuleiten, die die Implementierung und den Betrieb von Low-Code-Anwendungen in produzierenden Unternehmen unterstützen können.

Teilziel IV: Gestaltungsmaßnahmen für den Einsatz von Low-Code

Das finale Modell dieser Dissertation verfolgt das pragmatische Ziel, reale Sachverhalte zu gestalten (s. ZELEWSKI 2008, S. 24). Die Entwicklung dieses Modells wurde durch das Bestreben motiviert, Unternehmen konkrete, praxisorientierte Handlungsempfehlungen auf Grundlage der ermittelten wechselseitigen Wirkbeziehungen zwischen den Low-Code-Anwendungsfalltypen und den Low-Code-Regeln anzubieten. Ziel war es, Unternehmen zu befähigen, abhängig vom Anwendungsfalltyp, die relevanten Low-Code-Regeln mit gezielten Maßnahmen zu gestalten. Die leitende Forschungsfrage lautete:

Wie kann das Vorgehen zur anwendungsfallbasierten Definition eines Regelwerks für Low-Code in produzierenden Unternehmen gestaltet werden?

Basierend auf den Expert*inneninterviews konnten 60 Gestaltungsmaßnahmen abgeleitet werden. Diese Empfehlungen wurden den einzelnen Low-Code-Regeln zugeordnet und in Abhängigkeit des Low-Code-Anwendungsfalltyps differenziert. Dabei kann zwischen typenunabhängigen und typenspezifischen Low-Code-Regeln unterschieden werden. Das Ergebnis dieser methodischen Herangehensweise ist ein detaillierter Katalog von Maßnahmen, der Unternehmen eine gezielte Unterstützung bei der Implementierung und dem Betrieb von Low-Code-Anwendungen bietet. Das zweite

Ergebnis ist ein strukturiertes Vorgehen, um das Regelwerk in der Praxis anwendungsfallspezifisch anzuwenden und die bestehende IT-Governance innerhalb eines Unternehmens um die identifizierten Low-Code-Aspekte zu erweitern. Dieses abschließende Modell stellt somit eine wesentliche Ressource dar, um Low-Code-Anwendungen erfolgreich zu implementieren und zu betreiben und bietet eine solide Grundlage für den systematischen Einsatz von Low-Code in produzierenden Unternehmen.

10.2 Ausblick

Der in der Dissertation entwickelte Rahmen bietet vielfältige Ansatzpunkte für zukünftige Forschungsarbeiten sowie praktische Implikationen. Nachstehend erfolgt eine systematische Darstellung des Handlungsbedarfs.

Aus den Validierungsgesprächen wurde deutlich, dass sich in der Praxis eine Vermischung der Low-Code-Anwendungsfalltypen erkennen lässt. Zukünftige Forschungen sollten daher die in dieser Arbeit definierten Typen als Basis nutzen, um die Ursache der Vermischung weiter zu untersuchen und die Definition der Typen gegebenenfalls anpassen und deren Auswirkungen auf das Low-Code-Regelwerk zu untersuchen.

Darüber hinaus besteht Forschungsbedarf im Bereich des Low-Code-Regelwerks. Zur Identifikation der Low-Code-Regeln erfolgte eine Orientierung an dem COBIT-Kernmodell. Es ist notwendig, die identifizierten Regeln besonders im Bereich der Anwendungsentwicklung weitergehend zu untersuchen. Die verschiedenen Phasen des Softwareentwicklungs-Lebenszyklus werden im Regelwerk vereinfacht dargestellt. Mit zunehmender Komplexität der Low-Code-Anwendungen, bedarf es einer detaillierten Untersuchung des Entwicklungsprozesses.

Eine tiefergehende Untersuchung der Wirkbeziehungen zwischen den Low-Code-Anwendungsfalltypen und Regeln ist ebenfalls erforderlich. Die bisherigen qualitativen Expert*inneninterviews haben zwar Zusammenhänge aufgezeigt, jedoch keine Korrelationen geprüft. Es wird eine vertiefende quantitative Forschung empfohlen, möglicherweise durch ein quasi-experimentelles Design, um verschiedene Perspektiven zu integrieren und die externe Validität zu priorisieren.

Schließlich spiegeln die entwickelten Gestaltungsmaßnahmen den Zustand zum Zeitpunkt der Dissertation wider. Angesichts der rapiden technologischen Entwicklungen und des zunehmenden Einsatzes von KI ist eine Erweiterung der Maßnahmen zur Einhaltung des Regelwerks notwendig. Dadurch bleibt die Relevanz und Anwendbarkeit des Regelwerks auch in Zukunft erhalten und kann den praktischen Nutzen für Unternehmen sicherstellen, die Low-Code-Anwendungen einsetzen möchten.

11 Quellenverzeichnis

11.1 Publikationsverzeichnis

FRINGS, K.; SCHUH, G.: Description And Typification Of Low-Code Use Cases In Manufacturing Companies. In: IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD) (2023), S. 1–6.

11.2 Literaturverzeichnis

ACHARYA, B.; SAHU, P. K.: Software Development Life Cycle Models: A Review Paper. In: International Journal of Advanced Research in Engineering and Technology (IJARET) (2020), S. 169–176.

ADRIAN, B.; HINRICHSEN, S.; NIKOLENKO, A.: App Development via Low-Code Programming as Part of Modern Industrial Engineering Education. In: Advances in Human Factors and Systems Interaction. Hrsg.: I. L. Nunes. Advances in Intelligent Systems and Computing. Springer International Publishing, Cham, 2020a, S. 45–51.

ADRIAN, B.; HINRICHSEN, S.; SCHULZ, A.; VOß, E.: Low-Code-Programmierung als Ansatz zur Gestaltung bedarfsgerechter informatorischer Assistenzsysteme – eine Fallstudie. In: Informatorische Assistenzsysteme in der variantenreichen Montage - Theorie und Praxis (2020b), S. 173–186.

AL ALAMIN, M. A.; MALAKAR, S.; UDDIN, G.; AFROZ, S.; HAIDER, T. B.; IQBAL, A.: An Empirical Study of Developer Discussions on Low-Code Software Development Challenges. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) (2020), S. 46–57.

ALAZZAWI; ABDULBASIT, YAS, QAHTAN M.; RAHMATULLAH, B.: A Comprehensive Review of Software Development Life Cycle methodologies. Pros, Cons, and Future Directions. In: Iraqi Journal for Computer Science and Mathematics (2023), S. 173–190.

APEL, S.; KÄSTNER, C.: An Overview of Feature-Oriented Software Development. In: Journal of Object Technology (2009), S. 1–36.

ARNOLD, L.; JÖHNK, J.; VOGT, F.; URBACH, N.: A Taxonomy of Industrial IoT Platforms' Architectural Features. In: Innovation Through Information Systems (2021), S. 404–421.

AXELOS: ITIL Foundation. ITIL 4 Edition. PEOPLECERT INTERNATIONAL LTD. Cyprus, 2021.

BALZERT, H.: Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering. 3. Auflage. Spektrum Akademischer Verlag. Heidelberg, 2009.

Benac, R.; Mohd, T. K.: Recent Trends in Software Development: Low-Code Solutions. In: Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3. Hrsg.: K. Arai. Lecture Notes in Networks and Systems. Springer International Publishing, Cham, 2022, S. 525–533.

Berwander, J.; Hahn, U.: Interne Revision und Compliance. Springer Fachmedien Wiesbaden, 2020.

Biethan, J.; Rohrig, N.: Datenmanagement. In: Handbuch Wirtschaftsinformatik (1990), S. 737–755.

Binzer, B.; Winkler, T. J.: Democratizing Software Development: A Systematic Multivocal Literature Review and Research Agenda on Citizen Development. In: Software Business. Hrsg.: N. Carroll; A. Nguyen-Duc; X. Wang; V. Stray. Lecture Notes in Business Information Processing. Springer International Publishing, Cham, 2022, S. 244–259.

Bock, A. C.; Frank, U.: Low-Code Platform. In: Business & Information Systems Engineering 63 (2021) 6, S. 733–740.

Böhm, R.; Fuchs, E.: System-Entwicklung in der Wirtschaftsinformatik. 5. vdf, 2002.

Broy, M.; Kuhrmann, M.: Einführung in die Softwaretechnik. SpringerVieweg. Berlin, Heidelberg, 2021.

BSI-Standard 200-1: Managementsysteme für Informationssicherheit (ISMS). BSI. Bundesamt für Sicherheit in der Informationstechnik, 2017.

Cabot, J.: Positioning of the low-code movement within the field of model-driven engineering. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. Hrsg.: E. Guerra; L. Iovino. ACM, New York, NY, USA, 2020, S. 1–3.

Carroll, N.; Nguyen-Duc, A.; Wang, X.; Stray, V. (Hrsg.): Software Business. Lecture Notes in Business Information Processing. Springer International Publishing. Cham 2022.

Costabile, M. F.; Fogli, D.; Letondal, C.; Mussio, P.; Piccinno, A.: Domain-Expert Users and their Needs of Software Development. In: 2nd International Conference on Universal Access in Human-Computer Interaction (2003), S. 532–536.

Da Cruz, M. A. A.; Paula, H. T. L. de; Caputo, B. P. G.; Mafra, S. B.; Lorenz, P.; Rodrigues, J. J. P. C.: OLP—A RESTful Open Low-Code Platform. In: Future Internet 13 (2021) 10, S. 249.

Dalibor, M.; Heithoff, M.; Michael, J.; Netz, L.; Pfeiffer, J.; Rumpe, B.: Generating customized low-code development plat-forms for digital twins. In: Journal of Computer Languages (2022) 70, S. 101–117.

Degenhardt, D.: Low-Code-Plattformen und deren Ursprünge. In: Hochschule Karlsruhe (2020), S. 1–7.

DERN, G.: Integrationsmanagement in der Unternehmens-IT. Systemtheoretisch fundierte Empfehlungen zur Gestaltung von IT-Landschaft und IT-Organisation. 1. Aufl. Vieweg + Teubner in GWV Fachverlage. Wiesbaden, 2011.

DI RUSCIO, D.; KOLOVOS, D.; LARA, J. de; PIERANTONIO, A.; TISI, M.; WIMMER, M.: Low-code development and model-driven engineering: Two sides of the same coin? In: Software and Systems Modeling 21 (2022) 2, S. 437–446.

DÜNNEBACKE, D.: Beschreibung und Typisierung der IT-Unterstützung im Maschinen- und Anlagenbau. 1. Auflage. Apprimus Verlag, 2015.

EIGNER, M.; ROUBANOV, D.; ZAFIROV, R. (Hrsg.): Modellbasierte virtuelle Produktentwicklung. Springer Vieweg. Berlin, Heidelberg 2014.

EISEND, M.; KUß, A.: Grundlagen empirischer Forschung. Springer Fachmedien Wiesbaden. Wiesbaden, 2017.

ELSHAN, E.; DICKHAUT, E.; EBEL PHILIPP: An Investigation of Why Low Code Platforms Provide Answers and New Challenges. In: Proceedings of the 56th Hawaii International Conference on System Sciences 2023 (2023), S. 6159–6169.

ESCALONA, M.-J.; PARCUS DE KOCH, N.; ROSSI, G.: A Quantitative SWOT-TOWS Analysis for the Adoption of Model-Based Software Engineering. In: The Journal of Object Technology 21 (2022) 4, 4:1.

FETTKE, P.; LOOS, P. (Hrsg.): Entwicklung eines Bezugsrahmens zur Evaluierung von Referenzmodellen - Langfassung eines Beitrages. Johannes Gutenberg-Universität Mainz 2004.

FINK, A.; SCHNEIDEREIT, G.; VOß, S.: Grundlagen der Wirtschaftsinformatik. 2. Physica. Heidelberg, 2005.

FLEIß, J.: Paul Lazarsfelds typologische Methode und die Grounded Theory. Generierung und Qualität von Typologien. In: Österreichische Zeitschrift für Soziologie 35 (2010) 3, S. 3–18.

FREIRE, S.; RIOS, N.; MENDONÇA, M.; FALESSI, D.; SEAMAN, C.; IZURIETA, C.; SPÍNOLA, R. O.: Actions and impediments for technical debt prevention. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. Hrsg.: C.-C. Hung; T. Cerny; D. Shin; A. Bechini. ACM, New York, NY, USA, 2020, S. 1548–1555.

FRIEDLI, T.: Die Architektur von Kooperationen. Difo-Druck. Bamberg, 2000.

FRIEDRICHS, J.: Methoden empirischer Sozialforschung. 14. Springer Fachmedien Wiesbaden GmbH, 1990.

FRINGS, K.; SCHUH, G.: Description And Typification Of Low-Code Use Cases In Manufacturing Companies. In: IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD) (2023), S. 1–6.

GARTNER (Hrsg.): Definition of Citizen Developer, 2022. https://www.gartner.com/en/information-technology/glossary/citizen-developer (Link zuletzt geprüft: 21.06.2024).

GAULKE, M.: Praxiswissen COBIT. Grundlagen und praktische Anwendung in der Unternehmens-IT. Geeignet als Vorbereitung auf die ISACA-Prüfungen: COBIT Foundation, IT-Governance & IT-Compliance Practitioner, IT-Governance-Manager, IT-Compliance-Manager, CGEIT. 3rd ed. dpunkt.verlag. Heidelberg, 2019.

GÖTZEN, R. F.: Ordnungsrahmen für die softwarebasierte Automatisierung administrativer Prozesse. 1. Auflage. Apprimus Verlag, 2022.

GROßE-OETRINGHAUS, W. F.: Fertigungstypologie. unter dem Gesichtspunkt der Fertigungsablaufplanung. Duncker & Humblot, 1974.

GURUNG, G.; SHAH, R.; JAISWAL, D. P.: Software Development Life Cycle Models-A Comparative Study. In: International Journal of Scientific Research in Computer Science, Engineering and Information Technology (2020), S. 30–37.

HABERFELLNER, R.; FRICKE, E.; WECK, O. de; VÖSSNER, S.: Systems Engineering. Grundlagen und Anwendung. 13. orell füssli Verlag. Zürich, 2015.

HANSCHKE, I.: Enterprise Architecture Management - einfach und effektiv. Ein praktischer Leitfaden für die Einführung von EAM. 2. , überarbeitete Auflage. Hanser. München, 2016.

HENNESSY, S. D.; LAUER, G. D.; ZUNIC, N.; GERBER, B.; NELSON, A. C.: Data-centric security: Integrating data privacy and data security. In: IBM Journal of Research and Development 53 (2009) 2, S. 1–12.

HERRMANN, A.: Grundlagen der Anforderungsanalyse. Standardkonformes Requirements Engineering. Springer Vieweg. Wiesbaden, Heidelberg, 2022.

HESS, T.; MATT, C.; BENLIAN, A.; WIESBÖCK, F.: Options for Formulating a Digital Transformation Strategy. In: MIS Quarterly Executive 15 (2016) 2, S. 103-119.

HEUER, M.; KURTZ, C.; BÖHMANN, T.: Towards a Governance of Low-Code Development Platforms Using the Example of Microsoft PowerPlatform in a Multinational Company. In: Hawaii International Conference on System Sciences (HICSS) (2022), S. 6881–6890.

HEVNER, A. R.; MARCH, S. T.; PARK, J.; RAM, S.: Design Science in Information Systems Research. In: MIS Quarterly (2004), S. 75–105.

HIRZEL, M.: Low-Code Programming Models, 04.05.2022. http://arxiv.org/pdf/2205.02282v1 (Link zuletzt geprüft: 21.06.2024).

HOFFMANN, J.: Informationssystem-Architekturen produzierender Unternehmen bei software-definierten Plattformen. 1. Auflage. Apprimus Verlag, 2018.

HOLTSCHKE, B.; HEIER, H.; HUMMEL, T.: Quo vadis CIO? Springer Berlin Heidelberg. Berlin, Heidelberg, 2009.

HOOGSTEEN, D.; BORGMAN, H.: Empower the Workforce, Empower the Company? Citizen Development Adoption. In: Hawaii International Conference on System Sciences (2022), S. 4717–4726.

HUSSAIN, F.; HUSSAIN, R.; NOYE, B.; SHARIEH, S.: Enterprise API Security and GDPR Compliance: Design and Implementation Perspective. In: IT Professional 22 (2020) 5, S. 81–89.

IHIRWE, F.; DI RUSCIO, D.; MAZZINI, S.; PIERINI, P.; PIERANTONIO, A.: Low-code engineering for internet of things: a state of research. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (2020), S. 1–8.

IHIRWE, F.; INDAMUTSA, A.; DI RUSCIO, D.; MAZZINI, S.; PIERANTONIO, A.: Cloud-based modeling in IoT domain: a survey, open challenges and opportunities. In: 2021 ACM/IEEE International Conference (2021), S. 73–82.

IHO, S.; KREJCI, D.; MISSONIER, S.: Supporting Knowledge Integration with Low-Code Development Platforms. In: European Conference on Information Systems (2021), S. 1233–1242.

INDAMUTSA, A.; DI RUSCIO, D.; PIERANTONIO, A.: A Low-Code Development Environment to Orchestrate Model Management Services. In: Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems. Hrsg.: A. Dolgui; A. Bernard; D. Lemoine; G. von Cieminski; D. Romero. IFIP Advances in Information and Communication Technology. Springer International Publishing, Cham, 2021, S. 342–350.

ISACA: COBIT 2019 Rahmenwerk. Einführung und Methodik. ISACA. Schaumburg, 2020.

ISO 15704: Industrial automation systems - Requirements for enterprise-reference ar-chitectures and methodologies. ISO; ICS: 25.040.01. Institute of Electrical and Electronics Engineers, 2005.

ISO 14764: Software engineering--software life cycle processes--maintenance. ISO. Institute of Electrical and Electronics Engineers. New York, NY, 2006.

ISO 42010: Systems and software engineering — Architecture description. ISO. Institute of Electrical and Electronics Engineers, 2011.

ISO 27001: Information security, cybersecurity and privacy protection. ISO; ICS: 35.030. Institute of Electrical and Electronics Engineers, 2022.

ISO 27002: Information security, cybersecurity and privacy protection. ISO; ICS: 35.030. Institute of Electrical and Electronics Engineers, 2022.

ISO 38500: Information technology. ISO; ICS: 35.020. Institute of Electrical and Electronics Engineers, 2024.

JEDNASZEWSKI, M.: https://www.mendix.com/blog/understand-no-code-vs-low-code-development-tools/, 2024. https://www.mendix.com/blog/understand-no-code-vs-low-code-development-tools/ (Link zuletzt geprüft: 21.06.2024).

JOHANNSEN, W.; GOEKEN, M.: Referenzmodelle für IT-Governance. Methodische Unterstützung der Unternehmens-IT mit COBIT, ITIL & Co. 2., aktualisierte und erw. Aufl. dpunkt-Verl. Heidelberg, 2011.

JUHAS, G.; MOLNAR, L.; JUHASOVA, A.; ONDRISOVA, M.; MLADONICZKY, M.; KOVACIK, T.: Low-code platforms and languages: the future of software development. In: 20th International Conference on Emerging eLearning Technologies and Applications (ICETA) (2022), S. 286–293.

JUNG, H.: Allgemeine Betriebswirtschaftslehre. 4., korrigierte und aktualisierte Aufl. Oldenbourg. München, 2012.

JUSSEN, P.: Betriebskennlinien für industrielle Dienstleistungen. 1. Auflage. Apprimus Verlag, 2016.

KAIB, M.: Enterprise Application Integration. Grundlagen, Integrationsprodukte, Anwendungsbeispiele. Springer Fachmedien Wiesbaden GmbH, 2002.

KEMPER, A.; EICKLER, A.: Datenbanksysteme. Eine Einführung. 7. Oldenbourg. München, 2009.

KHORRAM, F.; MOTTU, J.-M.; SUNYÉ, G.: Challenges & opportunities in low-code testing. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. Hrsg.: E. Guerra; L. Iovino. ACM, New York, NY, USA, 2020, S. 1–10.

KLOTZ, M.; BARTONITZ, M.: Compliance. Definition, Anwendung und Nutzen der IT-Compliance. https://www.business-wissen.de/artikel/was-ist-it-compliance-und-wie-setzt-man-sie-um/ (Link zuletzt geprüft: 21.06.2024).

KLOTZ, M.; DORN, D.-W.: IT-Compliance — Begriff, Umfang und relevante Regelwerke. In: HMD Praxis der Wirtschaftsinformatik 45 (2008) 5, S. 5–14.

KLUGE, S.: Empirisch begründete Typenbildung. Zur Konstruktion von Typen und Typologien in der qualitativen Sozialforschung. Springer Fachmedien Wiesbaden GmbH, 1999.

KNOBLICH, H.: Betriebswirschaftliche Warentypologie. Springer, 1969.

KNOLL, M.; STRAHRINGER, S. (Hrsg.): IT-GRC-Management - Governance, Risk und Compliance. Grundlagen und Anwendungen. Praxis der Wirtschaftsinformatik. Springer Vieweg. Wiesbaden, Heidelberg 2017.

KNUTH, D.: The Art of Computer Programming. Volumes 1-4a-1. Addison-Wesley, 1997.

KÖNIGS, H.-P.: IT-Risikomanagement mit System. Praxisorientiertes Management von Informationssicherheits- und IT-Risiken. Springer Vieweg. Wiesbaden, 2017.

KRALLMANN, H.; BOBRIK, A.; LEVINA, O.: Systemanalyse im Unternehmen. Prozessorientierte Methoden der Wirtschaftsinformatik. 6., überarb. und erw. Aufl. Oldenbourg-Verl. München, 2013.

KRÄMER, J.: Mittelstand 2.0. Typabhängige Nutzungspotenziale von Social Media in mittelständischen Unternehmen. Springer Gabler, 2014.

KRCMAR, H.: Informationsmanagement. Springer Berlin Heidelberg. Berlin, Heidelberg, 2015.

KUBICEK, H.: Heuristische Bezugsrahmen und heuristisch angelegtes Forschungsdes-ign als Elemente einer Konstruktionsstrategie empirischer Forschung. In: Empirische und handlungstheoretische Forschungskonzeptionen (1977), S. 3-36.

KUMAR, M.; RASHID, E.: An Efficient Software Development Life cycle Model for Developing Software Project. In: International Journal of Education and Management Engineering 8 (2018) 6, S. 59–68.

LACKES, R.; SIEPERMANN, M.: IT-Governance. https://wirtschaftslexikon.gabler.de/definition/it-governance-53193/version-276288 (Link zuletzt geprüft: 21.06.2024).

LAGERSTEDT, R.: Using automated tests for communicating and verifying non-functional requirements. In: IEEE 1st International Workshop on Requirements Engineering and Testing (RET) (2014), S. 26–28.

LEAVITT, H. J.: Applied organizational change in industry. structural, technological and humanistic approaches. Carnegie Institute of Technology, Graduate School of Industrial Administration, 1962.

LEBENS, M.; FINNEGAN, R.; SORSEN, S.; SHAH, J.: Rise of the Citizen Developer. In: Muma Business Review 5 (2022), S. 101–111.

LEHNER, F.; WILDNER, S.; SCHOLZ, M.: Wirtschaftsinformatik. Eine Einführung. Hanser, 2008.

LELOUDAS, P. (Hrsg.): Introduction to Software Testing. Apress. Berkeley, CA 2023.

LETHBRIDGE, T. C.: Low-Code Is Often High-Code, So We Must Design Low-Code Platforms to Enable Proper Software Engineering. In: Leveraging Applications of Formal Methods, Verification and Validation. Hrsg.: T. Margaria; B. Steffen. Lecture Notes in Computer Science. Springer International Publishing, Cham, 2021, S. 202–212.

LUO, Y.; LIANG, P.; WANG, C.; SHAHIN, M.; ZHAN, J.: Characteristics and Challenges of Low-Code Development. In: Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Hrsg.: F. Lanubile. ACM, New York, NY, USA, 2021, S. 1–11.

MAHANTI, R.: Data Governance and Data Management. Contextualizing Data Governance Drivers, Technologies, and Tools. 1st ed. 2021. Springer Singapore; Imprint Springer. Singapore, 2021.

MALONE, T. W.; CROWSTON, K.: The interdisciplinary study of coordination. In: ACM Computing Surveys 26 (1994) 1, S. 87–119.

MAREIS, C.: Quadratisch praktisch gut. Zur Erfolgsgeschichte des morphologischen Kastens. In: Verpackungen des Wissens (2012), S. 109-121.

MATHIJSSEN, M.; OVEREEM, M.; JANSEN, S.: Source Data for the Focus Area Maturity Model for API Management, 21.07.2020. http://arxiv.org/pdf/2007.10611v7 (Link zuletzt geprüft: 21.06.2024).

MAYRING, P.: Qualitative Inhaltsanalyse. In: Texte verstehen : Konzepte, Methoden, Werkzeuge (1984), S. 159–175.

MAYRING, P.: Einführung in die qualitative Sozialforschung. Eine Anleitung zu qualitativem Denken. Beltz. Weinheim, 2002.

MCFARLAN, F. W.; MCKENNEY, J. L.; PYBURN, P.: The information archipelago - plotting a course. In: Harvard Business Review (1983), S. 145–156.

MCNIFF, J.; WHITEHEAD, J.: All you need to know about Action Research. An Introduction. Sage Publications, 2006.

MEIER, A.: Ziele und Aufgaben im Datenmanagement aus der Sicht des Praktikers. In: Wirtschaftsinformatik (1994), S. 455–464.

MESAGLIO, M.; HOTLE, M.: Pace-Layered Application Strategy and IT Organizational Design: How to Structure the Application Team for Success. In: Gartner (2016), S. 1–13.

METZGER, A.; POHL, K.: Software product line engineering and variability management: achievements and challenges. In: Future of Software Engineering Proceedings. Hrsg.: J. Herbsleb; M. B. Dwyer. ACM, New York, NY, USA, 2014, S. 70–84.

MEYER, M.; ZARNEKOW, R.; KOLBE, L. M.: IT-Governance. Begriff, Status quo und Bedeutung. In: Wirtschaftsinformatik 45 (2003) 4, S. 445-448.

MOSKAL, M.: NO-CODE APPLICATION DEVELOPMENT ON THE EXAMPLE OF LOGOTEC APP STUDIO PLATFORM. In: Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 11 (2021) 1, S. 54–57.

MUCHHAL, R.; SHARMA, M.; PAITHANKAR, K.: Usability Attributes and Their Mapping in Various Phases of Software Development Life Cycle. In: Emerging Trends in Expert Applications and Security. Hrsg.: V. S. Rathore; J. M. R. S. Tavares; V. Piuri; B. Surendiran. Lecture Notes in Networks and Systems. Springer Nature Singapore, Singapore, 2023, S. 153–162.

MURDOCH, R.: Robotic Process Automation. Guide To Building Software Robots, Automate Repetitive Tasks & Become An RPA Consultant. Eigenverlag, 2018.

NACHREINER, F. (Hrsg.): Grundlagen naturwissenschaftlicher Methodik in der Arbeitswissenschaft. Schäffer-Poeschel 1997.

NG'AMBI, D.: Empathy-driven Mobile App Aevelopment (MAD) without Coding. A Case of Citizen Developers. In: Critical Mobile Pedagogy (2020), 136-150.

NICKERSON, R. C.; VARSHNEY, U.; MUNTERMANN, J.: A method for taxonomy development and its application in information systems. In: European Journal of Information Systems 22 (2013) 3, S. 336–359.

NITHITHANATCHINNAPAT, B.; JOSHI, K. D.: A global view of what fixes information technology skills shortage: Panel data analyses of countries' human and technology resources. In: Journal of Global Business Insights 4 (2019) 1, S. 59–77.

ÖSTERLE, H.; BECKER, J.; FRANK, U.; HESS, T.; KARAGIANNIS, D.: Memorandum zur gestaltungsorientierten Wirtschaftsinformatik. In: Schmalenbach Journal of Business Research (2010), S. 664–672.

OTEYO, I. N.; PUPO, A. L. S.; ZAMAN, J.; KIMANI, S.; MEUTER, W. de; BOIX, E. G.: Building Smart Agriculture Applications Using Low-Code Tools: The Case for DisCoPar. In: 2021 IEEE AFRICON (2021), S. 1–6.

OVEREEM, M.; JANSEN, S.; MATHIJSSEN, M.: API Management Maturity of Low-Code Development Platforms. In: Enterprise, Business-Process and Information Systems Modeling. Hrsg.: A. Augusto; A. Gill; S. Nurcan; I. Reinhartz-Berger; R. Schmidt; J. Zdravkovic. Lecture Notes in Business Information Processing. Springer International Publishing, Cham, 2021, S. 380–394.

PASMORE, W. A.: Social Science Transformed. The Socio-Technical Perspective. In: Human Relations (1995), S. 1–21.

POHL, K.; RUPP, C.: Basiswissen Requirements Engineering. dpunkt verlag, 2015.

PRESSMAN, R. S.; MAXIM, B. R.: Software engineering. A practitioner's approach. Ninth edition. McGraw-Hill Education. New York, NY, 2020.

PRINZ, N.; HUBER, M.; RIEDINGER, C.; RENTROP, C.: Two Perspectives of Low-Code Development Platform Challenges – An Exploratory Study. In: PACIS 2022 Proceedings. (2022), S. 235–251.

PROBST, G. J.; GOMEZ, P.: Vernetztes Denken. 2. Gabler, 1991.

PÜSCHEL, L.; SCHLOTT, H.; RÖGLINGER, M.: What's in a Smart Thing? Development of a Multi-layer Taxonomy. In: Proceedings of the 37th International Conference (2016), S. 1-19.

RAHARJA, I. M. S.; SIAHAAN, D. O.: Classification of Non-Functional Requirements Using Fuzzy Similarity KNN Based on ISO / IEC 25010. In: l2th International Conference on Information & Communication Technology and System (ICTS) 2019 (2019), S. 264–269.

RAHIMI, N.; EASSA, F.; ELREFAEI, L.: An Ensemble Machine Learning Technique for Functional Requirement Classification. In: Symmetry 12 (2020) 10, S. 1601.

REDDY, S.; RUDRA, B.: Evaluation of Recurrent Neural Networks for Detecting Injections in API Requests. In: IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC) (2021), S. 936–941.

Rehäuser, J.; Krcmar, H. (Hrsg.): Wissensmanagement in Unternehmen. de Gruyter 1996.

Ritchey, T.: Fritz Zwicky, Morphologie and Policy Analysis. In: 16th EURO conference on operational analysis (1998), S. 1–12.

Ritchey, T.: General Morphological Analysis. A general method for non-quantified modelling. In: Swedish Morphological Society (2002), S. 1–10.

Rogozov, Y. I.; Lapshin, V. S.; Borovskaya, M. A.: Approach to the Implementation of Intelligent Low-Code Platforms. In: Proceedings of the Sixth International Scientific Conference “Intelligent Information Technologies for Industry” (IITI’22). Hrsg.: S. Kovalev; A. Sukhanov; I. Akperov; S. Ozdemir. Lecture Notes in Networks and Systems. Springer International Publishing, Cham, 2023, S. 42–50.

Rokis, K.; Kirikova, M.: Challenges of Low-Code/No-Code Software Development: A Literature Review. In: Perspectives in Business Informatics Research. Hrsg.: Ē. Nazaruka; K. Sandkuhl; U. Seigerroth. Lecture Notes in Business Information Processing. Springer International Publishing, Cham, 2022, S. 3–17.

Rüter, A.; Schröder, J.; Göldner, A.; Niebuhr, J.: IT-Governance in der Praxis. Springer Berlin Heidelberg. Berlin, Heidelberg, 2010.

Rymer, J.; Richardson, C.: The Forrester Wave: Low-Code Development Platforms, Q2 2016. The 14 Providers That Matter Most And How They Stack Up. https://www.forrester.com/report/The-Forrester-Wave-LowCode-Development-Platforms-Q2-2016/RES117623 (Link zuletzt geprüft: 21.06.2024).

Sahay, A.; Indamutsa, A.; Di Ruscio, D.; Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (2020), S. 171–178.

Salnitri, M.; Paja, E.; Giorgini, P.: Preserving Compliance with Security Requirements in Socio-Technical Systems. In: Communications in Computer and Information Science (2014), S. 49–61.

Sanchis, R.; García-Perales, Ó.; Fraile, F.; Poler, R.: Low-Code as Enabler of Digital Transformation in Manufacturing Industry. In: Applied Sciences 10 (2020) 1, S. 12.

Sawant, A. A.; Bacchelli, A.: fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage. In: Empirical Software Engineering 22 (2017) 3, S. 1348–1371.

Schaffry, A.: No-Code / Low-Code 2023. Die (R)evolution der Software-Entwicklung, 2023. https://shop.computerwoche.de/portal/studie-no-code-low-code-2023-pdf-download-direkt-im-shop-10750 (Link zuletzt geprüft: 05.03.2023).

SCHMIDT, A.; DORS, A.; KRETSCHMANN, C.; NESTLER, D.; MEISSNER, D.; UNTUCHT, G.; KARAMITROS, I.; ZILIAN, J.; KLOTZ, M.; KUNZEWITSCH, M.; DAX, O.; LOOS, P.; PURDER, S.: Leitfaden IT-Compliance. Grundlagen, Regelwerke, Umsetzung. ISACA Germany Chapter e.V., 2021.

SCHUH, G.; ANDERL, R.; GAUSEMEIER, J.; HOMPEL, M.; WAHLSTER, W. (Hrsg.): Industrie 4.0 Maturity Index. Die digitale Transformation von Unternehmen ge-stalten. Utz 2017.

SCHUH, G.; STICH, V.; HICKING, J.; WENGER, L.; ABBAS, M.; BENNING, J.; BREMER, M.; CLEMENS, F. (Hrsg.): Aachener Digital-Architecture-Management. Wegweiser zum digital vernetzten Unternehmen : Positionspapier. FIR-Edition Praxis; Bd. 13. FIR e.V. an der RWTH Aachen. Aachen 2020.

SCHULTE-ZURHAUSEN, M.: Organisation. Vahlen. München, 1995.

SCHÜTTE, R.: Grundsätze ordnungsmäßiger Referenzmodellierung. Gabler, 1998.

SCHWANINGER, M.: Model-based Management. A Cybernetic Concept. In: Systems (2015), S. 564-578.

SCHWARTZEL, T.: The impact of critical business data on organizations. In: AFRICAN JOURNAL OF BUSINESS MANAGEMENT 6 (2012) 26, 7705-7713.

SEAMAN, C.; GUO, Y. (Hrsg.): Advances in Computers. Chapter 2 - Measuring and Monitoring Technical Debt. 82. Academic Press / Elsevier Science & Technology 2011.

SEIBOLD, H.: T-Risikomanagement. Oldenbourg. München/Wien, 2006.

SIEGERS, J.: Gestaltung der interorganisationalen Zusammenarbeit mithilfe von Social Software. Apprimus Verlag. Aachen, 2016.

SMEETS, M.; ERHARD, R. U.; KAUßLER, T.: Robotic Process Automation (RPA) in der Finanzwirtschaft. Technologie – Implementierung – Erfolgsfaktoren für Entscheider und Anwender. Springer Gabler. Wiesbaden, Heidelberg, 2019.

SOMMERVILLE, I.: Software engineering. Tenth edition. Pearson. Boston, 2016.

STACHOWIAK, H.: Allgemeine Modelltheorie. Springer. Wien, 1976.

STAHL, T.; VÖLTER, M.: Model-driven software development. Technology, engineering, management. John Wiley. Chichester England, Hoboken NJ, 2006.

STRAHRINGER, S.: Modelle und Metamodellierung. In: Geschäftsprozessorientierte Systementwicklung. Von der Unternehmensarchitektur zum IT-System (2016), S. 11–24.

STYCZYNSKI, Z. A.; RUDION, K.; NAUMANN, A.: Einführung in Expertensysteme. Springer Berlin Heidelberg. Berlin, Heidelberg, 2017.

SUTTON, D.; SUTTON, M.: Wheels Within Wheels: a Development of Traditional Socio-Technical Thinking. In: Management Education and Development (1990), S. 122–132.

Szelągowski, M.; Lupeikiene, A.; Berniak-Woźny, J.: Drivers and Evolution Paths of BPMS: State-of-the-Art and Future Research Directions. In: Informatica (2022), S. 399–420.

Talesra, K.; G. S., N.: Low-Code Platform for Application Development. In: International Journal of Applied Engineering Research 16 (2021) 5, S. 346.

techconsult GmbH: Low-Code-/No-Code-Development Enabler der digitalen Transformation. Status quo und Planungen von Low-Code-/No-Code-Development-Plattformen in deutschen Unternehmen, 2021. https://www.smapone.com/no-code-studie/ (Link zuletzt geprüft: 05.03.2024).

Tiemeyer, E. (Hrsg.): Handbuch IT-Management. Konzepte, Methoden, Lösungen und Arbeitshilfen für die Praxis. 7. Hanser. München 2020.

Ullrich, C., Lata, T., Geyer-Klingeberg, J (Hrsg.): Celonis Studio – A Low-Code Development Platform for Citizen Developers. IEEE 2020.

Ulrich, H.: Die Betriebswirtschaftslehre als anwendungsorientierte Sozialwissenschaft. In: Management (1984), S. 168-199.

Ulrich, P.; Hill, W.: Wissenschaftstheoretische Grundlagen der Betriebswirtschaftslehre. In: Zeitschrift für Ausbildung und Hochschulkontakt (1976), S. 304–309.

Umar, M.; Khan, N. A.: Analyzing Non-Functional Requirements (NFRs) for software development. In: IEEE 2nd International Conference (2011), S. 675–678.

Umudova, S.: Analysis of Software Maintenance Phases. In: Nobel Internation Journal of Scientific Research (2019), S. 62-66.

Urbach, N.; Ahlemann, F.: IT Management in the Digital Age. A Roadmap for the IT Department of the Future. 1st ed. 2019. Springer International Publishing; Imprint: Springer. Cham, 2019.

Vester, F.: Die Kunst vernetzt zu denken. Ideen und Werkzeuge für einen neuen Umgang mit Komplexität. dtv, 2015.

Wang, J.; Qi, B.; Zhang, W.; Sun, H.: A Low-Code Development Framework for Constructing Industrial Apps. In: Computer Supported Cooperative Work and Social Computing. Hrsg.: Y. Sun; D. Liu; H. Liao; H. Fan; L. Gao. Communications in Computer and Information Science. Springer Singapore, Singapore, 2021, S. 237–250.

Waszkowski, R.: Low-code platform for automating business processes in manufacturing. In: IFAC-PapersOnLine 52 (2019) 10, S. 376–381.

Weill, P.; Ross, J. W.: IT governance. How top performers manage IT decision rights for superior results. Harvard Business School Press. Boston, 2004.

Welter, M.: Die Forschungsmethode der Typisierung. Charakteristika, Einsatzbereiche und praktische Anwendung. In: Informationen für Studium und Beruf (2006), S. 113–116.

WÖHE, G.; DÖHRING, U.; BRÖSEL, G.: Einführung in die allgemeine Betriebswirtschaftslehre. 26. Vahlen. München, 2016.

WONG, J.: The Importance of Citizen Development and Citizen IT, 2019. https://blogs.gartner.com/jason-wong/importance-citizen-development-citizen/ (Link zuletzt geprüft: 21.06.2024).

ZACHMAN, J. A.: A framework for information systems. In: IBM Systems Journal Reprint (1987), S. 454–470.

ZANGEMEISTER, C.: Nutzwertanalyse in der Systemtechnik. 4. Wittemansche Buchhandlung, 1976.

ZELEWSKI, S.: Betriebswirtschaftslehre. 4. Oldenbourg, 2008.

ZIEGLER, R.: Typologien und Klassifikationen. In: Soziologie (1973), S. 11–47.

ZWICKY, F.: Morphologische Astronomie. In: Physikalische Blätter 5 (1949) 1, S. 4–10.

Anhang

A.1 Analyse der COBIT-Kernmodell Ziele

Ziel	Beschreibung	Bewertung
EDM01	Governance-Rahmenwerk	Zu allgemein
EDM02	Mehrwerte	Implizit
EDM03	Risiko-Optimierung	Implizit
EDM04	Ressourcen	Aufnehmen
EDM05	Anspruchsgruppen	Aufnehmen
APO01	IT-Management-Rahmenwerk	Zu grob
APO02	Strategie	Nicht im Fokus
APO03	Unternehmensarchitektur	Aufnehmen
APO04	Innovation	Nicht im Fokus
APO05	Portfolio	Zu grob
APO06	Budget und Kosten	Nicht im Fokus
APO07	Personal	EDM04
APO08	Beziehungen	EDM05
APO09	Servicevereinbarungen	DSS01
APO10	Lieferanten	Nicht im Fokus
APO11	Qualität	Aufnehmen
APO12	Risiko	Implizit
APO13	Sicherheit	DSS05
APO14	Daten	Aufnehmen
BAI01	Programme	Zu allgemein
BAI02	Anforderungsdefinition	Aufnehmen
BAI03	Lösungserstellung	Aufnehmen
BAI04	Verfügbarkeit und Kapazität	EDM04
BAI05	Organisatorische Änderungen	EDM05
BAI06	IT-Änderungen	BAI02
BAI07	Umsetzung und Abnahme von IT-Änderungen	BAI03
BAI08	Wissen	Aufnehmen
BAI09	Betriebsmittel	Implizit
BAI10	Konfiguration	APO03
BAI11	Projekte	BAI02
DSS01	Betrieb	Aufnehmen
DSS02	Support	Aufnehmen
DSS03	Probleme	DSS02
DSS04	Kontinuität	DSS01
DSS05	Sicherheitsservices	Aufnehmen
DSS06	Geschäftsprozesskontrollen	APO11
MEA01	Leistung und Konformität	Zu allgemein
MEA02	Internes Kontrollsystem	Zu allgemein
MEA03	Compliance mit externen Anforderungen	Zu allgemein
MEA04	Assurance	Zu allgemein

A.2 Expert*inneninterview Anwendungsfalltyp I

Experte Operational Excellence und einem Masteranden eines deutschen Elektronikherstellers

Datum: 14. November 2022

Ort: Digital über Microsoft Teams

Experte 1 [00:02:50] Ich habe vorher Wirtschaftsingenieurwesen in Aachen studiert und habe dann meine Masterarbeit direkt hier geschrieben im Jahr 2020, eben zum Thema Power BI und habe dabei Dashboards erstellt. Das war für mich der erste Schritt in Richtung Low-Code-Plattform und bearbeite jetzt auch hauptsächlich alles rund um das Thema Kennzahlen, Kennzahlenvisualisierung, Digitalisierung und Prozessoptimierung.

Experte 2 [00:03:21] Ich bin hier im Unternehmen für meine Masterarbeit und beschäftige mich mit dem Shopfloor-Management und der Industrie 4.0. Ich bringe die Themen Shopfloor-Management, was ein altes Thema vom Prinzip ist, und münze das in die Industrie 4.0. Ich schaue, welche Stufen und welche Anforderungen sich im Shopfloor-Management ergeben und wie man das Ganze integrieren kann. Selbst bin ich, bevor ich hier war, schon vier Jahre bei einem Webhosting Unternehmen angestellt gewesen im Bereich Backend-Entwicklung. Deswegen bin ich eigentlich eher der typische Entwickler auf der Programmierebene. Ich finde aber Power Plattformen, oder generell Low-Code-Anwendungen, sehr interessant, was eben auch die Wirtschaftsinformatik ist, weil es die Schnittstellen aufweicht und man dabei auch Leute aus dem Business in die Programmierung hineinlassen kann.

Interviewer (FIR e.V.) [00:06:23] Vielleicht könnt ihr noch mal konkret sagen, wie genau eure Position lautet und was da genau eure Hauptaufgaben sind.

Experte 1 [00:06:40] Meine Position nennt sich Experte Operational Excellence aus der Abteilung Operational Excellence aus der Business Unit Laundry. Das heißt, wir beschäftigen uns mit der Wäschepflege, also Standort Gütersloh. Ich bin aber auch für die Standorte Uničov und Ksawerów, also Polen und Tschechien, teilweise, mit dabei. Meine Hauptaufgaben behandeln die Themen Shopfloor Management, also wie die Prozesse auf dem Shopfloor im Bereich Shopfloor Management ablaufen. Außerdem mit Themen der Digitalisierung, also konkret auch mit der Digitalisierung auf dem Shopfloor, bezüglich Power BI Dashboards, die dann für den Shopfloor genutzt werden. Ebenso generell Prozessoptimierung und Prozessdigitalisierung.

Experte 2 [00:07:29] Ich bin aus derselben Abteilung. Unsere Unterabteilung heißt Process Consulting and Digitalization. Im Prinzip geht es um Themen aus der Prozessoptimierung, Wertschöpfungsoptimierung oder um den Bereich Digitalisierung.

Interviewer (FIR e.V.) [00:08:06] Vielen Dank. Nun zum Ablauf, wie das Thema Low-Code, entstanden ist bei euch: Wart ihr in dem Prozess, als das Ganze aufgebaut wurde, bereits involviert, oder seid ihr in eure Stellen stattdessen im Anschluss gerutscht?

Experte 1 [00:08:20] Ich bin damals mit meiner Masterarbeit dort reingerutscht. Es stand schon vorher fest, dass wir das Programm Power BI nutzen sollten. Das war damals in meinen Anfängen. Ich bekomme es jetzt aber teilweise mit, wie wir ein Citizen Development aufbauen wollen. Ich bin da aber nicht selbsttätig dabei.

Experte 2 [00:08:42] Da ich jetzt erst drei Monate hier bin, waren das für mich schon geregelte Rahmenbedingungen. Wir sind aber abseits von diesen Themen bei der Suche nach anderen Plattformen, die den Funktionsumfang erweitern. Hierbei waren wir mit verschiedenen Anbietern, sei es zum Beispiel Siemens, im Austausch.

Interviewer (FIR e.V.) [00:09:12] Und ihr seid beide Programmierer? Sprich, ihr baut auch die jeweiligen Tools, die Low-Code-Anwendungen, bei euch selbst auf?

Experte 1 [00:09:21] Ja.

Interviewer (FIR e.V.) [00:09:23] Der nächste Schritt wäre, dass wir nun tiefer in den Einsatz von Low-Code bei euch im Unternehmen eintauchen. Was ist das Ziel, das mit eurer Arbeit, mit der Umsetzung von Low-Code-Anwendungen, erreicht werden soll?

Experte 1 [00:09:46] Zielt die Frage darauf ab, dass wir generell Low-Code-Anwendungen nutzen oder jetzt wirklich unsere ganz spezifischen Themen, und was wir mit diesen erreichen wollen?

Interviewer (FIR e.V.) [00:09:55] Also zunächst gerne auf eure spezifischen Themen, sprich, was möchtet ihr konkret umsetzen. Wenn ihr das Wissen habt, dann gerne auch danach in einem größeren Rahmen.

Experte 1 [00:10:10] Also meine Aufgabe ist Dashboards zu bauen für das Shopfloor Management. Das heißt, hierbei ist das Ziel, Daten digital bereitzustellen und manuellen Aufwand zu eliminieren. Also wir haben teilweise, beziehungsweise hatten noch, manuelle Tafeln, auf denen morgens Daten ausgedruckt oder angepinnt wurden. Und das Ziel ist nun, dies zu digitalisieren und diesen Aufwand zu vermeiden.

Experte 2 [00:10:43] Ich bin auch in einem Projekt, bei dem wir verschiedene Shopfloor Tafeln digitalisieren. Ich würde sagen, der Digitalisierungsanspruch ist ein konkretes Ziel. Wobei wir wirklich digitalisieren, anstatt dass wir neue Software dazu holen. Dies als erster Schritt, bei dem die Digitalisierung an erster Stelle steht, nämlich dass wir viele analoge Prozesse eliminieren können.

Experte 1 [00:11:14] Ich denke etwas in die Richtung diverser Power Apps, die haben nicht wir selbst gebaut, aber die gibt es bei uns in der Umgebung. Ich würde sagen, dass die alle das Ziel haben, Daten zu digitalisieren und diese dann schneller zur Verfügung zu stellen, als es mit den manuellen Prozessen bisher funktionierte.

Interviewer (FIR e.V.) [00:11:34] Wenn wir jetzt den Blick erweitern, vielleicht auch auf das gesamte Unternehmen oder innerhalb eures Arbeitsfokus, in dem ihr euch auskennt: Ist dies auch die allgemeine Aufgabe von Low-Code bei euch? Sprich, das Visualisieren und Digitalisieren von Daten und Prozessen?

Experte 1 [00:11:54] Ja, man könnte das sogar noch erweitern. Um Prozessoptimierung, beziehungsweise um Prozesse, zu verschlanken.

Interviewer (FIR e.V.) [00:12:03] Aber es ist schon so, dass die Daten erst einmal nur visualisiert werden. Sprich, die Software an sich verschlankt nicht unmittelbar den Prozess, sondern es geht nur um das Verständnis darüber, was verschlankt werden kann. Also die Verschlankung, die passiert nicht durch die Software selbst, sondern ermöglicht lediglich, dass Daten besser aufbereitet werden können, transparenter gestaltet werden können, um dann darauf basierend Entscheidungen zu treffen, die wiederum Prozesse optimieren? Konkret: Optimiert die Software selbst oder ermöglicht die Software eine Optimierung?

Experte 2 [00:12:53] Ich würde dem Zweiten zustimmen. Die Software ermöglicht die Optimierung.

Experte 1 [00:13:00] Das geht ein bisschen in die Richtung der Masterarbeit, dass wir aktuell mehr bei dem Stand sind, dass wir wirklich nur Visualisieren und Analysen ermöglichen. Aber, dass wir jetzt schon Schritte weiterdenken, wie wir damit bereits im Vorfeld Probleme erkennen können.

Experte 2 [00:13:20] Wir überlassen die Entscheidung natürlich an den meisten Stellen immer noch den Mitarbeitern, hinsichtlich Decision Support Systems ist das jetzt nicht angedacht. Wir ermöglichen nur die Verschlankung dadurch, dass wir den Prozess digitalisieren. Aber die Verschlankung an sich passiert dann natürlich darüber, dass wir das Ganze besser nutzen können. Sei es dadurch, dass wir Daten transparenter, echtzeitfähig oder aktueller darstellen können.

Interviewer (FIR e.V.) [00:13:56] Wurde vor der Low-Code-Lösung dieses Problem bereits adressiert? Beziehungsweise, wie wurden denn die Daten vorher, die existiert haben, verwendet? Waren solche Prozessoptimierungen bereits möglich?

Experte 1 [00:14:17] Wenn wir uns die Bereiche des Shopfloor Managements näher anschauen, wir haben dort diese manuellen Tafeln. Das heißt, da ist der aktuelle Prozess, dass sich die Anwender*innen aus diversen Systemen die relevanten Daten manuell herausziehen, ausdrucken und anpinnen. Als ich damals hierhin gekommen bin, gab es allerdings auch schon Dashboards. Die waren dann aber von der Zentral-IT gehostet. Das heißt, das war nichts, was man ohne große Programmierkenntnisse selbst entwickeln konnte. Und genau das ist jetzt ein Ziel, nämlich dass wir solche Projekte in die Fachabteilungen an Mitarbeiter wie mich, die nicht komplexen Code programmieren können, dafür aber Low-Code verstehen, weitergeben können.

Interviewer (FIR e.V.) [00:15:00] Nochmal zwei Fragen zurückgesprungen: Es existiert also zum einen die Prozessoptimierung als Ziel. Aber zusätzlich, so wie ich das verstanden habe, dass man in den Abteilungen auch eigenständiger und schneller neue Lösungen umsetzen kann.

Experte 2 [00:15:16] Genau. Die Beschleunigung der Digitalisierung ist dabei ein zentraler Aspekt.

Experte 1 [00:15:22] Ja, stimmt. Denn es geht auch um Citizen Developer, nämlich, dass neue Themen in die Fachabteilungen reingegeben werden.

Interviewer (FIR e.V.) [00:15:32] Sind denn nach der Low-Code-Einführung diese Probleme gelöst worden? Vorher war der Status quo: Es kann nur von Entwicklern entwickelt werden, und das nur auf einer hohen Flugebene. Wahrscheinlich hat dann der Kontakt zu den Mitarbeitern auch nicht stattgefunden, die das Tool nutzten oder umsetzten. Ist das jetzt anders? Seid ihr im direkten Kontakt mit denen, die das Tool verwenden?

Experte 1 [00:16:02] Ja. Wir haben jetzt eine sehr regelmäßige Feedbackschleife bei den Dashboards. Natürlich bei einigen mehr, bei anderen weniger. Aber es kommt die Rückmeldung, oder ich fordere sie dann ein.

Interviewer (FIR e.V.) [00:16:17] Zu den ganz frühen Zeiten, als euer Unternehmen sich dafür entschieden hat, Low-Code einzusetzen. Gab es damals gewisse Kriterien, nach denen entschieden wurde, dass ihr genau diese Daten oder diesen Prozess optimieren möchtet?

Experte 1 [00:16:52] Ich weiß nicht, was die Grundsatzentscheidung war. Das, was ich weiß, ist, dass wir zum Thema Shopfloor Management anfangs, oder sogar noch vor meiner Zeit, überlegt haben, eine Lösung einzuführen. Es gibt digitale Tools, bei denen auch Dashboards mit integriert sind. Und ich weiß, dass es damals hieß: „Wir probieren es einfach selbst auf die günstige Art und Weise aus, weil Power BI auch ein Microsoft Programm ist.“ Ich glaube, das war so ein Teil, der mit eingespielt hat. Aber mehr kann ich dazu nicht sagen.

Interviewer (FIR e.V.) [00:17:28] Zum Verständnis: Es war nicht so, dass ihr euch aus einem ganz bestimmten Grund für Power BI entschieden habt, sondern es war eher im Sinne von „low hanging fruits“.

Experte 1 [00:17:53] Ich würde sagen ja, aber ich kann es nicht genau sagen.

Interviewer (FIR e.V.) [00:17:58] Nun die abschließende Frage hinsichtlich der Zukunftsaussichten: Ihr seid aktuell dabei zu digitalisieren, Dashboards zu bauen. Gibt es für euch eine Zeit danach, wie Low-Code weiter ausgerollt werden kann? Oder welche Pläne ihr für die Zukunft kennt?

Experte 2 [00:18:33] Also hinsichtlich Citizen Developer, dass wir vielleicht noch mehr Leute in die Entwicklung hineinbringen können und dadurch vielleicht noch stärker im Business die Digitalisierung fördern können?

Interviewer (FIR e.V.) [00:18:53] Zum Beispiel. Aber auch, ob ihr schon wisst, was es für andere Möglichkeiten gibt, solche Tools für euch zu nutzen. Ihr baut gerade Dashboards zur Visualisierung von Daten, aber vielleicht gibt es auch andere Möglichkeiten wie Mini-Apps oder sonstiges, die bald vielleicht auch von euch entwickelt werden sollen?

Experte 2 [00:19:11] Hinsichtlich meiner Masterarbeit versuchen wir durch die Digitalisierung und Low-Code-Anwendungen in das Shopfloor Management einige Thematiken zu integrieren, die wir im Moment jetzt weniger haben. Sei es, dass wir horizontale Vernetzung mit integrieren, dass wir verschiedene Produktionsbereiche besser

vernetzen und dadurch hoffentlich die Produktion besser aufeinander abstimmen können und die Wertschöpfungskette besser optimieren können. Aber vielleicht auch, dass wir sagen, wir bauen auf diesen Daten, die wir aktuell sammeln und dadurch, dass wir digitale Lösungen gebaut haben, jetzt auch noch mehr Daten reinbekommen, was wir vielleicht vorher nur in analoger Weise irgendwie verfügbar hatten. Also, dass wir die Datengrundlagen, die wir aufgebaut haben, durch Low-Code-Anwendung stärker auswerten und damit versuchen, prädiktive Lösungen zu bauen, um die Entscheidungsfindung am Shopfloor zu optimieren.

Interviewer (FIR e.V.) [00:20:25] Das heißt, diese Prozessoptimierung, nämlich Daten zu visualisieren, und transparenter zu gestalten, noch weiter zu optimieren, weil ein Fluss von Daten aus der Optimierung auch wieder zurückkommt. Also das, was ihr gerade habt, noch mal auf eine höhere Ebene bringen und nicht eine komplett neue Low-Code-Struktur bei euch aufzubauen?

Experte 2 [00:20:57] Das würde ich so bestätigen. Wir bauen auf das, was wir bereits haben, auf. Versuchen es also zu erweitern, anstatt …

Interviewer (FIR e.V.) [00:21:04] … eine komplett neue Perspektive einzunehmen. Verstanden.

Experte 1 [00:21:09] Was man noch als konkrete Maßnahme ergänzen kann: Unsere Idee ist, dass wir Maßnahmen aus dem Shopfloor Management, also aus den Kennzahlen, die das Dashboard zeigt, ableiten können. Sprich, dass wir diese Maßnahmen im besten Fall sogar in dem Dashboard notieren können und dass wir eine Power BI Lösung kreieren, um Eskalationskaskaden einzubauen, mit denen man dann eine Maßnahme direkt zu dem Verantwortlichen rüberschicken kann und diese in dem entsprechenden Dashboard wieder aufploppt.

A.3 Expert*inneninterview Anwendungsfalltyp II

IT Effizienz Manager und Digital Operations Manager eines Schweizer Lebensmittelherstellers
Datum: 17. November 2022

Ort: Digital über Microsoft Teams

Experte 1 [00:00:11] Ich bin Anfang des Jahres als IT Effizienz Manager eingestellt worden und bin für die Erstellung von Low-Code, aber auch für die Schulung der ganzen Fachabteilungen, zuständig. Ich bin jetzt nicht für eine spezielle Fachabteilung zuständig, sondern soll im Unternehmenskosmos überall hin und schauen, wo ich Effizienzen schaffen kann. Und das Wichtige hierbei ist, dass wir momentan in einer Phase sind, in der wir das Ganze aufbauen. Wir werden momentan eine Governance einrichten, die eben zentral solche Anfragen bedient, Lösungen entwickelt und langsam den Mitarbeiter dazu heranführt, selbst Low-Code auszuführen.

Experte 2 [00:01:08] Ich bin Digital Operations Manager und verantworte im Bereich Produktion, Logistik, Technik die Digitalstrategie. Das heißt, ich überlege, welche Systeme brauchen wir zur Unterstützung unserer Aufgaben. Ich bin aber auch Ansprechpartner im Fachbereich, also interner Berater, was die Planung von Projekten und deren Umsetzung und Controlling angeht. Ich arbeite stark mit der IT zusammen, an der Stelle für die Low-Code-Anwendung. Man muss dazu sagen, wir sind hier insbesondere im Microsoft Umfeld, was wir aktuell stark nutzen. Das ist der erste Ansatz. Und wir sind dabei, das ganzheitlich aufzubauen, denn aktuell ist das noch nicht der Fall. Meine Hauptaufgaben sind Projekte identifizieren, steuern und im Rahmen der Gesamt-Roadmap auf das Zielbild hin arbeiten zu lassen. Wobei die Erstellung des Zielbilds für den Bereich Operations in meiner Verantwortung liegt, das Zielbild im Gesamtunternehmen werden wir gemeinsam entwickeln. Wir haben hier also den Bottom-Up Ansatz, dass wir eigenständig unser Zielbild entwickeln können.

Interviewer (FIR e.V.) [00:02:43] Vielen Dank. Du hattest gesagt, dass du die Low-Code-Anwendungen erstellst. Sprich, du bist also der Entwickler?

Experte 1 [00:03:20] Tatsächlich ist das Tool schon vorhanden. Jeder kann sofort den Code erzeugen. Das machen wir mithilfe von einem Microsoft Tool namens Power Automate. Jeder ist durch die Lizenzierung dazu bereit, eigene Lösungen zu entwickeln. Und was ich mache, ist den Mitarbeitern aufzuzeigen, wie man das machen kann. Ich erstelle mithilfe der Tools, nehme die Anforderung auf und setze es um. Das könnte aber theoretisch jeder machen, derjenige braucht aber natürlich das Wissen dazu.

Experte 2 [00:03:53] Nichtsdestotrotz gibt es auch in Fachabteilungen findige Leute, die eigenständig Anwendungen erstellen.

Experte 1 [00:04:05] Genau, das ist eine der Voraussetzungen, einmal wie man das Tool letztendlich bedienen kann, aber auch dieses Logikverständnis mitzubringen. Auch wenn man keine IT-Ausbildung hat, kann man, wie zum Beispiel über Excel, Funktion in eine Abfolge bringen. Wie muss was hin, welche Aktion muss wann

ausgeführt werden. Und genau das kann die Fachabteilung am besten bewerten, wie ihre Prozesse funktionieren. Und das kann ich nur in Zusammenarbeit mit der Fachabteilung auch erstellen. Aber wenn sie so weit sind, können sie auch selber ihren Low-Code einbringen. Ich bin der Opener dafür, dass die Fachabteilung überhaupt den Blick darauf hat, die einzelnen Tools über Low-Code verbinden zu können. Stand jetzt wird das oft manuell getätigt. Das heißt, wir haben eine Excel-Datei, die wird fertig gemacht, dann wird sie per Mail verschickt. Oder in Teams wird eine Nachricht geschickt mit „ich bin fertig“. Das könnte man über Low-Code zusammenführen und automatisieren.

Interviewer (FIR e.V.) [00:05:10] Seid ihr denn dann auch an dem Prozess beteiligt, welche Anwendungen konkret umgesetzt werden?

Experte 2 [00:05:25] Aktuell noch nicht. Aktuell läuft das alles unter dem Radar, und das ist für uns ein großes rotes Tuch, weil so eine Anwendungslandschaft entsteht, die weder von der IT noch von den Fachbereichen wirklich kontrolliert wird. Deswegen sind wir an der Governance dran. Also bei meiner Einführungsphase, ich bin erst seit Mitte des Jahres hier, habe ich auch einige selbst erstellte Apps kennengelernt. Die wurden mir vorgestellt als Teil der Digitalisierung. Das Problem ist, dass an der Stelle die Anforderungen nicht komplett kommuniziert wurden, sondern Sachen einfach direkt umgesetzt werden und somit entsteht eine riesige Schattenseite. Man muss dazu wissen: das Werk ist groß. Das heißt, wir haben knapp 2000 Leute in der Produktion, die in sechs Abteilungen sind und jede Abteilung zwischen 150 und 750 Mitarbeitern hat. Und dort werden dann über Abschlussarbeiter, Projektmitarbeiter, Hilfskräfte zum Teil solche Anwendungen programmiert, zum Teil durch Abteilungsleiter selbst, die sehr findig sind und da die Zeit zu finden. Das wird aber alles nicht wirklich richtig kommentiert, es unterliegt keiner Governance, es gibt keine Struktur. Andere Abteilungen kriegen das dann rüber gespielt „hey, kannst du auch nutzen“, aber niemand hat sich gedacht, wie integriert sich das wirklich in die IT-Landschaft, geschweige denn gibt es diese Funktion in der IT-Landschaft schon. Und daher wollen wir das über die Governance angehen und die Leute erstmal dazu befähigen, die Anwendung vernünftig aufzubauen.

Experte 1 [00:07:19] Was von unseren Händen ausgeht, das haben wir natürlich im Blick und haben das in den entsprechenden Listen gepflegt. Das können wir natürlich verantworten, nämlich was genau erstellt wird. Aber wie mein Kollege ausgeführt hat, ist schon eine Blackbox vorhanden.

Interviewer (FIR e.V.) [00:07:38] Wir haben jetzt viel über aktuelle Problematiken gesprochen. Was mich jetzt, auch für den zweiten Block der Fragen, interessieren würde: was sind denn die Ziele für euch und die Projekte? Bei euch in der Struktur, aber auch von den Anwendungen ausgehend?

Experte 2 [00:08:04] Wir haben aktuell zwei Stränge von Low-Code-Anwendungen. Wir haben einmal den Strang rund um die Office-Welt, der hauptsächlich durch all die Guerilla-Anwender, die mit Low-Code bereits jetzt schon losgelegt haben, die damit Sachen lösen, die durch die Standardsoftware, wir haben hier SAP hauptsächlich,

nicht abgedeckt werden können. Das Ziel ist es, insbesondere mit den Office-Anwendungen Sachen zu automatisieren, Workflows zu schaffen, die vor allem weit weg von SAP sind, was auch in vielen Bereichen aktuell nicht genutzt wird. Wobei wir dann aber die Sachen steuern können, Personen unabhängig machen, Prozesse beschleunigen dadurch, dass wir menschliche Handlungen wie einer Fertigmeldung „hey, ich habe die Excel bearbeitet", oder „hey, ich habe dieses Formular bearbeitet, bitte kümmere du dich jetzt um das auszudrucken und kommunizieren", digitalisieren. Ich denke da an den Anwendungsfall, beispielsweise bei unserem Training. Jemand hat den Wunsch für eine Weiterbildungsmaßnahme, die nicht im Standard-Portfolio ist, und das gilt als Anfrage rüber zusenden. Und dann kümmert sich ein Bereich darum, dieses Training zu buchen, zu schauen, welche gibt es denn. Und dieses dann für den Kollegen zu buchen und ihm dann zu sagen „hey, wir haben Termin für dich, passt das? Viel Spaß beim Training".

Der zweite Strang, der ist aktuell noch sehr gering. Wenn wir an SAP denken, das wird auch in Zukunft unser Hauptsystem sein, da wollen wir zum Standard hin. Das heißt, wir wollen den SAP-Kern eigentlich unangetastet lassen. In den letzten Jahren wurde vor allem über Eigen-Programmierung und Z-Transaktionen das System ausgeweitet. Wir haben das hier am Standort über die letzten 30 Jahre irgendwann auf die Spitze getrieben und aktuell ist der Code so unübersichtlich, dass wir uns nicht mal externe Hilfe dazu holen können, um Sachen anzupassen. Das soll in Zukunft nicht geschehen. Das heißt, wir wollen den Kern so unbelastet wie möglich lassen und über Low-Code-Anwendungen die Anforderungen aus den Fachbereichen an Vereinfachungen von Transaktionen knüpfen. Das System ist alles andere als nutzerfreundlich, auch wenn das immer angepriesen wird. Auch die neueste Version wird nicht unbedingt nutzerfreundlich sein. Vor allem, wenn man im Standard ist, sind es sehr viele Klicks und Transaktionen, die man hintereinander aufrufen wird. Das wollen wir über Low-Code-Anwendungen verschlanken. Das heißt, wir schaffen damit Workflows, Vereinfachungen, Benutzeroberflächen. Der Kern bleibt unangetastet. Darauf kommt dann die Low-Code-Anwendung, die vielleicht nur noch eine Maske darstellt, im Hintergrund aber drei Masken nacheinander mit den eingegebenen Infos füllt. Und das ist eigentlich das Ziel. Wir wollen Standardsoftware nutzen und über Low-Code die Vereinfachung für den Nutzer entsprechend unserer Workflows schaffen vor dem Gedanken, sich den wandelnden Anforderungen der Märkte besser stellen zu können. Wenn wir ein Standard-SAP-System haben, sind wir heute noch Lagerfertiger, können aber auch in Zukunft das Geschäft auf Auftragsfertigung umstellen oder Veränderungen in der Gruppenstruktur schneller umsetzen. Das heißt, es könnte auch sein, dass gar nicht mehr im Markt spezifisch in Deutschland produziert wird, sondern immer für das Intercompany. Wir denken da beispielsweise an die Vorwerk Gruppe. Vorwerk ist so aufgeteilt, dass es eine Vertriebsgesellschaft gibt, dann den Direktvertrieb von Kobolten, ein paar kleine Retail-Geschäfte. Dahinter gibt es drei Produktionsstandorte, die die gesamte Gruppe beliefern. So etwas könnte es möglicherweise auch hier in diesem Konzern geben, aber das wissen wir alles nicht. Mit unserem aktuellen SAP könnten wir diese Transformation nicht mitgehen. Das würde uns vor riesige Aufwände stellen, vor

riesige Herausforderung stellen, in der Abarbeitung. So viele Leute können wir gar nicht anschaffen, um in unseren aktuellen Prozessen solche geänderten Geschäftsmodelle umzusetzen. Und wir wissen nicht, wie sich das im Lebensmittelhandel weiterentwickelt, was die Anforderungen der Nutzer sind, gerade mit der alten Käuferschicht. Die Käufer sind von uns, so wie wir wissen, sind deutlich älter. Das wird sich wandeln und die Anforderungen an die Produkte werden sich ändern. Und das heißt auch, wir haben hier intern Änderungen, was Produktion angeht. Das können wir aktuell alles nicht abfangen über unser SAP. Und deswegen brauchen wir ein schnell wandelndes Unternehmen. Low-Code-Anwendungen bieten für uns die Möglichkeit, auch eigenständig Vereinfachungen an Workflows vorzunehmen. Und dahin wird sich die Zukunft entwickeln, wie wir das mit der Zentralisierung machen. Wer das anlegen darf, das wissen wir noch nicht, das überlegen wir uns jetzt im ersten Schritt der Governance. Und wie wir das dann demokratisieren, als eine mögliche Richtung, oder ob wir das weiterhin zentral halten, müssen wir schauen. Da müssen wir aber auch im Laufe der Zeit dann gegebenenfalls noch mal die Richtung anpassen.

Interviewer (FIR e.V.) [00:13:35] Es geht also zum einen darum, aktuell noch manuelle Prozesse zu automatisieren, sodass keine papier-basierten Aufgaben mehr existieren, sondern diese automatisiert und digitalisiert werden. Es geht aber auch darum, die IT-Landschaft, im Sinne des SAP, zu erweitern. Und zwar so, dass ihr einen Lock-In Effekt vermeiden möchtet, dass ihr autarker werden könnt, dass ihr neue Ideen und Erweiterungen der Landschaft nicht mehr zentral steuern wollt, sondern aus den Abteilungen heraus.

Experte 1 [00:14:50] Es soll schon zentral von unserer IT gemanagt werden. Es geht primär darum, dass wir mithilfe von Low-Code diese Prozesse, die wir sonst über Eigen-Programmierung abfackeln, damit ablösen können. Das ist das Ziel. Das, was nicht im Standard ist, möchten wir gerne mit Low-Code ablösen.

Experte 2 [00:15:22] Genau. Was man dazu sagen kann: Die Begriffe Autark und Abgabe in die Fachbereiche ist nicht das primäre Ziel, sondern es ist eine Vereinfachung der IT-Aufgaben. Wir haben eine sehr schlanke IT. Das wird sich wahrscheinlich über die nächsten Jahre auch leider nicht ändern können. Die IT ist deutlich zu klein für die Größe des Werks. Was wir damit schaffen wollen, ist eine Arbeitserleichterung der IT, weil wir glauben, dass es leichter zu handhaben ist und die Einarbeitung schneller vonstattengeht.

Experte 1 [00:15:57] Hinsichtlich Autark, das möchten wir vor allem in diesem ersten Punkt bringen. Dass die Abteilung für sich selbst Automatisierung im Office-Bereich durchführen können, ohne damit an die IT zu gehen „könnt ihr das bitte im SAP entwickeln?“. Sondern, dass sie sich selbst mit Low-Code-Lösungen schaffen können.

Experte 2 [00:16:25] Das ist richtig. Und wenn wir an die größeren Anwendungen rund um SAP denken, wollen wir uns weniger abhängig von den Eigen-Programmierungen machen, um Lösungen oder Services und Dienstleistungen besser am Markt einkaufen zu können. Wir haben aktuell eine SAP-Anwendung in der Instandhaltung. Da haben wir das SAP-Modul in den letzten vier Jahren neu eingeführt, haben dort

größtenteils den Standard. Das ist aber nicht wirklich nutzerfreundlich für unsere Instandhaltung, die vorher alles auf Papier zurückgemeldet hat. Was haben wir gemacht? Wir sind mit der Low-Code-Plattform „Neptune“ und einem Dienstleister hingegangen und haben in Fiori Optik, dem neuen SAP, eine vereinfachte Maske erstellt, die im Hintergrund mehrere Masken bedient und die Eingabe deutlich vereinfacht. Wir denken gerade über einen weiteren Fall der Anwendung dieser Low-Code-Plattform nach. Dabei sind wir aber nicht unbedingt aktuell die Entwickler dieser Applikation, sondern kaufen die Dienstleistung ein. Vor dem Hintergrund: Wir haben zum Teil das Know-How, es würde aber zu lange dauern und unsere IT ist aktuell noch mit dem Betrieb des SAP betreut und kann das aktuell nicht machen. Was wir dadurch erreichen wollen, ist, dass unsere IT nicht mehr mit dem reinen Betrieb, der Wartung und der Pflege unserer Transaktionen betreut ist, sondern vor allem Prozessoptimierung durchführen kann, indem es Low-Code-Anwendungen bereitstellt, pflegt und verbessert und gegebenenfalls Konfigurationen am Kern vornimmt, die aber alle im Standard bleiben.

Interviewer (FIR e.V.) [00:18:31] In welchen Abteilungen oder Unternehmensbereichen schult ihr gerade und in welchen ist als Low-Code-Thema aktuell am brisantesten?

Experte 1 [00:18:46] Ich würde sagen vor allem im Verwaltungsbereich. Dadurch, dass wir täglich mit Excel-Dateien hantieren, in irgendeiner Form austauschen, ist da natürlich, „low hanging fruits“ und es lassen sich schnell die Vorteile sehen, was mit Low-Code möglich ist, bei dem man ganz einfach die Tools miteinander verbinden kann. Aber sowohl auch im Produktionsbereich, vor allem in den Leads, bei denen sich Produktionsleiter kümmern, wie das Personal genau fungiert, sprich das Management, und dies durch Office-Werkzeuge bedienen. Auch die profitieren davon. Man muss dazu sagen, wir sind am Anfang alles publik zu machen, denn es gibt mittlerweile Low-Code, den wir auch einsetzen können. Bei euch im Produktionsbereich, wird das schon vermehrt eingesetzt. Wir hatten am Anfang gesprochen über diese Blackbox. Es gibt bereits Leute, die mehr Know-How mitbringen und die haben schon solche Tools im Einsatz.

Experte 2 [00:20:19] Ich glaube am stärksten nutzt es der Bereich aktuell, in dem ich bin, also der Bereich Operation, Planung, Logistik, Technik, Produktion. Und dabei insbesondere im Rahmen der Produktion. Ich weiß, dass darüber einige Checklisten digital geführt werden. Die Produktion macht regelmäßig Rundgänge, zum Beispiel bei denen Themen nachgehalten werden. Ich weiß, dass Lean-Vorgehen in solchen Anwendungen abgebildet werden. Das Problem ist, dass die Produktion großen Bedarf an Digitalisierung hat. Wir aber die Standardsoftware nicht so schnell nachschießen können und eigentlich zu wenig Leute für größere Projekte haben. Daraufhin wurde losgelegt, Sachen selbst zu machen. Das ist hier leider ein Vorgehen, was seit Jahren publik ist, weshalb wir sehr viel Provisorien schaffen, die Ewigkeiten Bestand haben. Aber ich glaube, geschult wird hier am wenigsten, was meiner Meinung nach auch am sinnvollsten ist, denn sonst würde hier ein riesiger Wildwuchs entstehen.

Interviewer (FIR e.V.) [00:21:35] Das heißt, es sind administrative Themen, die Zettelwirtschaft zu digitalisieren. Du hast gerade von Lean gesprochen, da denke ich spontan an Kanban-Karten, die über Low-Code einfacher verwaltet werden können?

Experte 2 [00:21:52] Nein, es ist nicht bei Kanban, also nicht in der Materialflusssteuerung, denn da ist man direkt im SAP-Umfeld. Bei Lean ist es bei uns der Fall, dass wir sogenannte Tech-Karten haben. Das heißt, wenn jemandem in der Produktion, ein Mitarbeiter oder eine Mitarbeiterin, Sachen auffallen, also eine Maschine nicht sauber ist, Sachen fehlen, Sachen instandgesetzt werden müssen, dann wird eine solche Karte geschrieben und die wird dann abgearbeitet. Das wird aktuell über eine Anwendung schon digitalisiert, und zwar auch in größerem Umfang. Ich glaube, das nutzen mittlerweile alle Abteilungen. Es ist aus einer Abteilung rausgekommen, dass dort die Maschinenbediener Sachen aufschreiben können, die ihnen auffallen, die wieder Instand gesetzt werden sollten. Und darüber wird dann Richtung Technik kommuniziert, also unsere Instandhaltung und Maschinentechnik, aber auch in Richtung Produktion selbst.

Experte 1 [00:22:59] Du hattest am Ende die Frage zu den Schulungen, richtig?

Interviewer (FIR e.V.) [00:23:09] Genau, in welchen Abteilungen ihr euch hauptsächlich aufhaltet, weil dort vermutlich der Ort ist, an dem Low-Code am ehesten umgesetzt wird.

Experte 1 [00:23:21] Mit den Schulungen sind wir gerade in der Verwaltung, das Thema publik zu machen. Dass es möglich ist, dass Ideen überhaupt formuliert werden können. Das werden wir aber so überführen, dass sie nicht, wie in der Produktion, direkt Lösungen angehen und diese dann an uns vorbei gehen. Wir haben direkt in den Schulungen gesagt, bitte informiert uns über die Lösungen, damit wir euch den Support geben können.

Interviewer (FIR e.V.) [00:24:02] Das heißt, es ist nicht so, dass ihr das Produktionssystem mit Low-Code erweitern wollt? Es geht nicht darum, zum Beispiel, Maschinenprogramme zu bauen, sondern um Administration in der Produktion.

Experte 2 [00:24:32] Ja genau, aktuell wird Administration vereinfacht. Die Mitarbeiter nutzen dort die Office-Umgebung. Ich kann mir vorstellen, dass sie, wenn sie die Zeit dazu haben, auch anfangen, tatsächlich Produktionssteuerung damit zu machen. Das will ich verhindern, weil dann entsteht ein riesiger Wildwuchs. Man muss dazu wissen, dass bei uns im Unternehmen insgesamt 86 Nationalitäten arbeiten, in der Produktion müssten es auf jeden Fall über 50 sein. Wenn wir da deutsche Anwendungen schreiben, dann verlieren wir einen Großteil der Belegschaft. Es wird in der Produktion auf Zetteln meistens in vier Sprachen geschrieben. Das kriegen wir wahrscheinlich über die Low-Code-Anwendung nicht gut abgebildet, aber es ist nur eine Frage der Zeit, bis das passiert. Aber bis dahin sollte eigentlich ein System laufen, dass das kann. Aktuell werden eher administrative Sachen umgesetzt und Reporting vereinfacht.

Interviewer (FIR e.V.) [00:25:44] Warum habt ihr euch genau für diese Low-Code-Umsetzung entschieden? War es über die Microsoft-Welt für euch am einfachsten zu lösen?

Experte 1 [00:26:11] Genau. Ich denke mal, der wichtigste Grund ist viel schneller zu reagieren, auch auf die Business-Anforderungen und nicht mehr über das klassische „wir kippen ein", „wir müssen erst Ressourcen beschaffen, dann muss das entwickelt werden". Sondern, dass wir mit Low-Code die Lösung direkt in das Business reinschieben können und davon profitieren. Und das ist einer der großen Antreiber.

Experte 2 [00:26:42] Warum die Leute sich für die Lösung entschieden haben, ist, weil sie verfügbar ist für die Leute, weil sie eine gute Dokumentation dazu im Internet finden und weil sie sich mit Office auskennen. Und deswegen nutzen sie die Software. Ich sprach davon, dass es aktuell die Überlegung gibt, eine zweite Applikation mit einer Low-Code-Anwendung zu erstellen, in einem richtigen Projekt eingebettet. Aus dem Grund sucht man gerade noch eine Low-Code-Plattform aus. Dort sind dann andere Entscheidungskriterien dabei. Da wird dann aber auch entschieden: Nutzen wir die, die wir schon haben? Nutzen wir eine Zusätzliche? Wer kann programmieren? Wer kann das intern verwalten? Was sind Funktionsumfänge? Und auch da ist Einfachheit der Bedienung und Veränderungsfähigkeit ein großer Punkt. Da werden wir aber auch mit externen Entwicklern zusammenarbeiten, die für uns diese Low-Code-Anwendungen aufsetzen werden. Dort haben wir durch das Projektteam eigenständig schon Kriterien definiert, was den Preis, Fixkosten oder Einmalkosten, laufende Kosten, Bedienbarkeit und Umsetzung der gewünschten Lösung in dieser Umgebung ist. Diese befindet sich auch in der SAP-Umwelt, war vorher einprogrammiert, fest und starr, auch in der SAP-Programmiersprache. Das wollen wir diesmal nicht machen.

Interviewer (FIR e.V.) [00:28:29] Kannst du nennen, welche Plattformen zur Wahl stehen?

Experte 2 [00:29:05] Neptune oder Mobisys ist es als Plattform.

Experte 1 [00:29:08] Im Office Bereich wäre es Power Automate. Es gibt aber auch reichliche Alternativen, die man sich anschauen könnte. Die haben wir momentan aber nicht im Einsatz.

Interviewer (FIR e.V.) [00:29:26] Kommen wir zu der letzten Frage, und zwar in die Zukunft geblickt. Der nächste Schritt wäre also, projektspezifischer Low-Code-Anwendungen bereitzustellen. Seht ihr da noch mehr für euch in der Zukunft als Potenzial?

Experte 1 [00:29:51] Definitiv. Wir haben jetzt schon Anwendungsfälle oder Projekte, die in der Pipeline sind, die umgesetzt werden sollen. Da gibt es jetzt schon welche und die haben gerade erst angefangen mit der Vermarktung. Wenn das präsent ist in den Köpfen, dass es eher „wünsch dir was" ist, dann wird das glaube ich deutlich mehr. Und da muss man wirklich schauen, wie man das Ganze priorisiert mithilfe einer Governance.

Interviewer (FIR e.V.) [00:30:16] Kannst du solche Projekte nennen?

Experte 1 [00:30:24] Projekte im Rahmen der verschiedenen Fachabteilungen, das ist beispielsweise im HR-Bereich, dort wird in Zukunft erst eine ganz große Lösung angeschafft. In der Zwischenzeit werden wir mit Low-Code eine Zwischenlösung bauen, die aber trotzdem viel mitbringt und Ersparnisse einbringt. Aber auch andere Fachabteilungen, die zentral auf digitale Genehmigungen abzielen. Dass man eine zentrale Plattform hat. Eine, die man normalerweise über Mailing hat, wird dann digital auf eine zentrale Plattform gehoben und über Low-Code verbunden. Das sind die ersten Projekte.

Interviewer (FIR e.V.) [00:31:19] Das heißt, Zeit überbrücken. Bevor man eine neue Software einführt eine Übergangslösung zu schaffen, die dann vielleicht 80% schon erreicht?

Experte 1 [00:31:31] Genau. Hauptsache das Ziel, die Effizienz zu erreichen. Dass die Zeit, die man in Low-Code reinsteckt, bis dann das große Tool diese Erleichterung bringt, gleich ist. Dann lohnt sich auch diese Einführung. Das ist einer der Nebenaspekte, die man mit betrachtet.

Experte 2 [00:32:02] Generell planen wir weitere Anwendungen mit Low-Code. Wir setzen, wie schon gesagt, die Governance auf, um Anforderungen zu sammeln. Große Anforderungen an Veränderungen bekomme ich schon mit. Kleinere Anforderungen, die wollen wir abfangen und uns dann überlegen: „Kriegen wir das über Low-Code abgebildet?“. Ich sehe die Chance darin, Standardsysteme zu nutzen, die Commodity bleiben. Wenn wir Daten sammeln und in den verschiedenen Fachabteilungen den Datenzugriff gewähren können, dann brauchen wir auch Low-Code-Anwendungen, was die Datenanalyse angeht. Wir sprachen jetzt hier vor allem von Prozess Low-Code. Aber auch Low-Code-Analytics wird in Zukunft für uns enorm wichtig sein. Das heißt, wir werden da eine einfachere BI Lösung haben, über die wir Auswertungen machen können. Und das wird auch irgendwann so weit gehen, dass wir sagen, wir machen Analytics Modelle, die über Statistik hinausgehen, bis hin zur Modellgenerierung. Da sprechen wir aber dann von sehr ferner Zukunft.

Experte 1 [00:33:45] Ich würde das gerne in zwei Kategorien packen. Die persönliche Produktivität, bei der der Mitarbeiter seine Arbeit automatisiert und die Kategorie Systeme, bei der dann Plattformen geschaffen werden oder bei einer SAP-Eigenentwicklung abgewickelt werden. Das sind die zwei großen Sparten, die wir sehen. In der ersten Sparte sehen wir Microsoft und den Office-Bereich, in dem wir das Ganze auch mehr autark haben möchten, bei der sich Mitarbeiter selbst über einen Self-Service Automatisierung bauen. Und über den anderen Weg möchten wir gerne weiterhin die Hand drüber halten, bei dem wir in SAP alles bereitstellen, aber dann auch darüber, dass wir wieder zurück zum Kern gehen, auch Support von extern heranholen können bei Bedarf. Das sind so die zwei Themenfelder, die wir dann in diese Richtung entwickeln möchten.

A.4 Expert*inneninterview Anwendungsfalltyp III

Experte Team Lead Production IT eines deutschen Automobilherstellers

Datum: 15. November 2022

Ort: Digital über Microsoft Teams

Experte [00:01:00] Meine Position nennt sich Team Lead Production IT. In der Aufgabe und in der Rolle allgemein, bin ich zuständig für digitale Lösungen in der Produktion. Ich bin für die Implementierung und Weiterentwicklung der Standardapplikationen zuständig. Angefangen von ERP über MES und das WMS. Darüber hinaus ist es meine Aufgabe, enterprise-übergreifende Prozesse in den Systemen abzubilden. Wir haben das als Enterprise Data Model bezeichnet, sodass wir losgelöst und in einer Art Metaebene die Datenflüsse entlang der Kern-Business-Prozesse darstellen und da einen Single Source of Truth erreichen. Aber meine Aufgabe besteht nicht nur darin, die Standardapplikationen zu managen und weiterzuentwickeln, sondern auch gezielt diese an den richtigen Stellen zu erweitern. Und da habe ich eine Architekten-Aufgabe und bin daran beteiligt, die Lösungen oder die Technologien auszuwählen, um die Anforderungen aus den Fachbereichen abzubilden.

Interviewer (FIR e.V.) [00:02:47] Das bedeutet, dass du auch tatsächlich in dem Entscheidungsprozess involviert bist, welche Low-Code-Anwendung entsprechende genutzt werden?

Experte [00:02:56] Richtig. Wir haben eine Art Systemlandschaft, die unsere Core-Architektur darstellt. Also wir versuchen schon den Ansatz zu verfolgen, möglichst viel in Standardapplikationen auch umzusetzen. Aber es gibt aufgrund von den Fähigkeiten der Applikationen, aber auch von wirtschaftlichen Gesichtspunkten, immer wieder Überlegungen, Dinge zu erweitern. Nämlich durch Drittanwendungen oder mit Low-Code-Applikationen ganz zielgerichtet im kleinen Umfang.

Interviewer (FIR e.V.) [00:03:34] Vorab zur Klärung: Entwickelst du auch selbst oder bist du nur für das Aufsetzen dieser Tools zuständig?

Experte [00:03:43] Also prozessual für den Entwicklungsprozess bin ich auch mit in der Verantwortung, weil wir die Regeln definieren, wie entwickelt wird. Teilweise kann es vorkommen, dass ich selbst implementiere, aber in der Regel sind das meine Teammitglieder. Noch sind wir nicht so weit, dass es wirklich in den Fachbereichen passiert. Aber auch das wäre perspektivisch möglich, unter Anleitung meines Bereichs oder meines Teams, und wir definieren dann die Spielregeln.

Interviewer (FIR e.V.) [00:04:21] Dann möchte ich einsteigen, konkreter auf das Low-Code-Thema zu kommen. Von der hohen Flughöhe ausgehend: Was ist konkret das Ziel bei euch, warum setzt ihr Low-Code ein? Du hattest bereits das MES genannt, das eine Basis war und das jetzt erweitert werden soll, das automatisiert werden soll. War das auch das Hauptziel, was in diesem Projekt für dich an oberster Stelle stand?

Experte [00:04:54] Ja, ein wichtiger Erfolgsfaktor ist es, wandlungsfähig zu sein. Und Wandlung heißt auch immer Anpassung der Systeme. In der Regel, wie auch bei uns,

wie wir am Anfang gestartet sind, mit einem Team, das aus Projektmanager oder Projektleiter besteht, in dem eigentlich nur die Anforderungen aufgenommen werden und dann von externen Leuten implementieren lassen. Dann ist die Wandlungsfähigkeit dadurch limitiert, dass es a) Geld kostet und b) Zeit braucht, weil die externen Ressourcen erst eingeplant werden müssen. Die ganze Kette wird dadurch deutlich länger. Deswegen war von Anfang an, und das war ein Stückweit der Grund, weshalb wir auch mit diesem MES-Anbieter zusammenarbeiten, das Ziel, irgendwann selbstständig intern Dinge anpassen zu können. Und da haben wir nie über wirkliches Coding gesprochen, sondern dass wir die Oberfläche, die der Bediener am Ende sieht, dass wir diese etwas editieren können oder auch die Sequenz von logischen Folgen, Entscheidungen, Eingaben, die der Mitarbeiter macht, etwas editieren. Von daher ist Wandlungsfähigkeit dabei der Hauptfaktor.

Interviewer (FIR e.V.) [00:06:20] Sprich Wandlungsfähigkeit und die Erweiterung von einer bereits bestehenden Software, Customization. Dass ihr das selbst vornehmen könnt und es nicht vom Anbieter in einem Loop mit euch entwickelt werden muss.

Experte [00:06:34] Richtig.

Interviewer (FIR e.V.) [00:06:37] Gab es vorher den Fall, dass Anpassungswünsche oder Anforderungen durch einen Entwickler programmiert wurden? Oder habt ihr stattdessen das System schlicht so gelassen, wie es war?

Experte [00:07:00] In den meisten Fällen haben wir gesagt, dass die Anforderungen des Fachbereichs legitim sind. Wir haben jetzt nicht eine große Analyse gemacht, ob es da einen Business-Case dahinter gibt, oder was ist, wenn wir es nicht tun. Sondern wir haben gesagt es ist legitim, es würde sehr viel Sinn ergeben. Und dann haben wir das so entsprechend angefragt.

Interviewer (FIR e.V.) [00:07:22] Okay. Du hattest MES erwähnt. Sprich, diese Anwendung läuft auf dem Shopfloor in der Produktion. Das ist der Rahmen, der für das Programm gesetzt ist?

Experte [00:07:38] Genau, wir können uns darauf fokussieren. Wir haben sicher auch andere Anwendungsfälle, aber alles, was auf dem Shopfloor eingesetzt wird, also Apps, die dem Mitarbeiter an die Hand gegeben werden, damit er besser durch seine Prozesse geführt wird oder damit wir die Informationen sammeln, die während der Arbeit der Mitarbeiter entstehen. Das heißt Rückmeldungen am Montageprozess oder Erfassung von Ergebnissen bei der Qualitätsprüfung, das sind solche Beispiele.

Interviewer (FIR e.V.) [00:08:16] Vielleicht können wir konkret darauf eingehen. Es wäre interessant zu wissen: Was nutzt ihr überhaupt für ein Tool und welche Apps oder Erweiterungen sind das? Auch dahingehend, welche bis jetzt am erfolgreichsten waren. Kurz: Was sind das für konkrete Aufgaben, die ihr bearbeitet?

Experte [00:08:40] Nehmen wir die Montagestation. Der Mitarbeiter hat ein Touch Display zur Verfügung. Bei uns wird das Fahrzeug noch wenig automatisiert montiert, sondern es sind wirklich Mitarbeiter. Das heißt, ein Mitarbeiter bekommt eine Liste an Prozessen, die er abzuarbeiten hat. Über dieses Display werden ihm Anweisungen zur

Verfügung gestellt und er wird durch den Prozess geleitet. Und ein klassisches Thema ist der Einsatz von Schraubtechnologie, also der Mitarbeiter hat ein entsprechendes Schraubwerkzeug. Das Werkzeug wird mit bestimmten Parametern gestartet und dann gibt es eine Rückmeldung. Und je nachdem, wie die Rückmeldung aussieht, muss der Mitarbeiter darauf reagieren können. Sagen wir mal, die Verschraubung war nicht in Ordnung. Dann ist immer die Frage, warum war sie nicht in Ordnung? Kann man sie reparieren oder müssen wir das ganze Produkt ausschleusen in den Nacharbeitsbereich, um es dort wieder zu fixen? Und gerade diese Logik, entlang dieses Workflows: Was passiert, wenn folgendes Ereignis eintritt, welche Information möchte ich bereitstellen? Und und und… da sind wir jetzt mittlerweile in der Lage, selbst anzupassen.

Interviewer (FIR e.V.) [00:10:18] Und das läuft alles über PSI?

Experte [00:10:30] Richtig, PSI ist der Hersteller, der Anbieter. MES ist das Produkt. Die Technologie, die dabei verwendet wird, ist eine Workflow Engine, die von Camunda kommt.

Interviewer (FIR e.V.) [00:10:52] Sprich über diese App implementiert ihr Lösungen in PSI oder wird das von PSI selbst angeboten, also die Möglichkeit der Low-Code-Funktionen?

Experte [00:11:22] Weil es Low-Code ist, können wir nur das implementieren, was uns zur Verfügung steht an Bausteinen. Und diese Logikbausteine, zum Beispiel „aktiviere den Schrauber mit einem Programm“, der hat Inputparameter und würde dann eine Funktion ausführen. Der Baustein, der muss natürlich vorab entwickelt werden. Das ist dann schon Coding und das müssen wir beauftragen, nämlich dass wir so einen Art Baustein bekommen, um etwas ansteuern zu können. Aber dann können wir diesen Baustein auch frei verwenden.

Interviewer (FIR e.V.) [00:12:00] Dieser Baustein wird von PSI entwickelt?

Experte [00:12:03] Genau. Das heißt, wir haben eine Bibliothek an Standardbausteinen, die wir von PSI entwickelt bekommen haben und damit ist unser Spielraum abgesteckt. Den können wir natürlich erweitern, dafür müssen wir aber PSI beauftragen.

Interviewer (FIR e.V.) [00:12:30] Warum habt ihr euch für diesen Weg entschieden? So wie ich es verstanden habe, habt ihr euch deshalb entschieden, weil das die einzige Möglichkeit war, die bestehende IT-Landschaft zu erweitern.

Experte [00:12:58] Ja, das ist richtig. Warum haben wir uns dafür entschieden? Zunächst ist die Entscheidung eine Weile her. Ich weiß nicht, ob ich sie heute wieder so treffen würde. Ich war auch nicht beteiligt an der Entscheidung, dieses Tool einzuführen. Aber mittlerweile muss ich sagen, haben wir dadurch wirklich eine Art Flexibilität erhalten, die uns auch für die Zukunft recht gut aufstellt. Aber der Vorteil von dieser Art Lösung ist: Wir haben einen MES-Anbieter, der die Anwendung MES schon komplett durchgedacht hat. Das heißt, es kommen mit einem Datenmodell Informationen, die typischerweise in der Produktion benötigt werden oder auch dokumentiert werden müssen. Dafür gibt es vorgedachte Speicherplätze und Datenwege. Und wenn wir jetzt Funktionen anfragen, sagen wir mal wir, wir wollen jetzt nicht nur einen Schrauber

bedienen, sondern auch eine Roboteranlage, dann müssen wir die Anforderungen nur relativ grob gehalten spezifizieren und sagen wir möchten den Roboter mit einem Programm versorgen und wir möchten ein Ergebnis haben. Aber was wir dann damit machen und wann wir den im Gesamtprozess aufrufen, da haben wir viel Spielraum und können das, ich nenne es mal „on-the-fly“, immer noch verändern. Wir müssen nicht alles, in Form von einem sehr spezifischen Lastenheft, vorab festhalten, sondern können später relativ einfach noch korrigieren. Das ist der Vorteil, den wir hier haben.

Interviewer (FIR e.V.) [00:14:48] Wenn du im Nachhinein sagst, vielleicht hättest du damals anders entschieden: Gibt es denn für die Zukunft Pläne, in Form neuer Schnittstellen, eine neue Software zu implementieren oder eine Erweiterungsmöglichkeit hinzuzufügen? Was sind die Pläne für die Zukunft, um die Low-Code-Landschaft zu erweitern?

Experte [00:15:13] Was bei uns noch ein Problem ist, ist wirklich die Gestaltung des Front-End, also das, was der Mitarbeiter sieht, wo er die klickbaren Elemente findet. Hierbei sind wir sehr eingeschränkt in den Elementen, die für uns nutzbar sind. Wir haben eine Lösung gefunden, aber die ist ganz und gar nicht Low-Code.

Interviewer (FIR e.V.) [00:15:46] Kannst du sagen, was das bedeutet? Eine neue Software, die diese Möglichkeit erweitert?

Experte [00:15:54] Diese Workflow Engine, die stellt im Prinzip die Information zur Verfügung. Es gibt Datenpunkte, die können aufgegriffen werden und wir haben jetzt einen Weg gefunden, wie wir bei einem Front-End mit React, das ist die Programmiersprache, Informationen aufgreifen und darstellen können. Das heißt, wir nutzen im Prinzip wiederum diese Logik des Workflows, aber haben obendrein ein selbst programmiertes Front-End. Was aber die Front-End Gestaltung angeht, da auch in Richtung Low-Code zu gehen, wäre schick. Was auf jeden Fall ein Vorteil für uns wäre, wenn die Bibliothek dieser Code Bausteine, die wir für die Workflow Modellierung nutzen, übersichtlicher und vollständiger für uns wäre. Wir sind dahingehend oft limitiert und denken es wäre gut, wenn wir noch mehr hätten. Es wäre auch gut, wenn wir noch mehr Standard-Schnittstellentechnologien verwenden könnten, sprich mehr generische Bausteine, denn die sind oft sehr spezifisch.

Interviewer (FIR e.V.) [00:17:34] Ist das denn aus deiner Perspektive mit Low-Code umsetzbar? Oder ist das stattdessen abhängig von PSI, dass diese euch die notwendigen Optionen zur Verfügung stellen?

Experte [00:17:50] Doch, da ist schon was umsetzbar. Aber es ist vielleicht dann schon Low-Code. Es ist Expert*innenwissen nötig, um es richtig umsetzen zu können. Ebenso brauchen wir Verständnis über Schnittstellentechnologien, Datenbanken, Datenbank-Modellierungen, Datenmodelle. Aber man muss jetzt nicht wirklich Zeilen an Code schreiben können, um das einsetzen zu können. Deswegen glaube ich schon, dass es ein Anwendungsfall ist.

A.5 Expert*inneninterview Wirkbeziehungen I1

Regel	Merkmal	Zitat
Anforderungs-definition	Anpassungs-möglichkeit	Wenn man Lösungen konzeptioniert, sollte man als erstes schauen, kann die Anforderung mit bestehenden Bausteinen erfüllt werden.
Anforderungs-definition	Geschäftskritikalität	Je kritischer die Anwendung ist, desto wichtiger ist es, dass Anforderungen sauber definiert sind.
Anforderungs-definition	Nutzung	Abteilungsinterne Anwendungen, können die Abteilungen gut organisieren, das muss auch nicht ein hochformalisierter Prozess sein. Aber abteilungsübergreifend und Standardübergreifend definitiv.
Anforderungs-definition	Schnittstellen	Die Schnittstellen spielen bei der Lösungsentwicklung eine so zentrale Rolle, dass man ganz klar identifizieren muss, ob alle notwendigen Schnittstellen entlang des Gesamtprozesses vorhanden sind oder ob der Bedarf besteht, konfigurierbare oder wirklich individuelle Schnittstellen zu implementieren?
Anspruchs-gruppen	Programmier-fähigkeiten	Der Austausch über die verteilten Teams ist immer wichtig. Auch bei erfahrenen Programmierern ist dies notwendig, weil sich die Low-Code-Anwendungen hinzu besser modularisierten Lösungen entwickeln sollen. Allerdings würde man den Austausch zwischen nur erfahrenen Entwicklern anders angehen, als wenn auch weniger erfahrene Leute mit reinkommen, weil die eher nochmal eine andere Perspektive haben.
Betrieb	Geschäftskritikalität	Also ganz kurz, also wenn es unkritisch ist, glaube ich, muss man nur drauf achten, dass, wenn man was oder das wenn was entwickelt wird, das auch klar kommuniziert ist. Was wird dadurch gelöst? Was wird vielleicht auch abgelöst? Das ist einfach im klaren Kommunikationsprozess gibt.
Betrieb	Geschäftskritikalität	Bei geschäftskritischen Teilen muss man klar definieren, wann wird etwas verändert. Und auch diese Frage kompatibel Kompatibilitätsthemen muss ich vielleicht andere Themen gleichzeitig noch machen?
Betrieb	Lebensdauer	Also in dem Zyklus kurz heißt, dass er dann einfach das ist dann das ganz kurze Betriebsdauer, sozusagen ne, das war fast gar nicht wahr, schlimm hat beides eine Einfluss, also plus ganz normal.
Betrieb	Nutzung	Das Gleiche können wir auch so bei der Nutzung übernehmen. (Bei wirklich geschäftskritischen Teilen muss man ganz klar definieren, wann wird etwas verändert. Und auch diese Frage kompatibel Kompatibilitätsthemen muss ich vielleicht andere Themen gleichzeitig noch machen?)
Betrieb	Nutzung	Das Gleiche können wir auch so bei der Nutzung übernehmen. (Also ganz kurz, also wenn es unkritisch ist, glaube ich, muss man nur drauf achten, dass, wenn man was oder das wenn was entwickelt wird, das auch klar kommuniziert ist. Was wird dadurch gelöst? Was wird vielleicht auch abgelöst? Das ist einfach im klaren Kommunikationsprozess gibt.)
Betrieb	Schnittstellen	Aber ich Kompatibilitätsthemen wie kommuniziere ich das? Muss ich, wenn ich dann was live nehme, sozusagen irgendwo in Config Files noch mal anfassen? Ein Schritt beim Deployment, dass man diese Konfiguration vielleicht

		mit deployed oder mit anpasst und ist es bei individualisiert noch mal mehr ein Thema als bei konfigurierbar.
Daten	Geschäftskritikalität	Also ich würde auch bei unkritischen ein Regelwerk für Daten muss jetzt nicht nichts Wildes sein, aber auch das hat immer n Einfluss drauf.
Daten	Geschäftskritikalität	Und bei kritisch und Geschäftskritisch. Ich muss noch mal hervorheben, Wenn man da besonders darauf achtet, wer da wer sieht, welche Daten, wer darf die verändern? Ja, genau.
Daten	Nutzung	Man muss erstmal definieren, welche Daten zur Verfügung stehen, um zu bewerten, dass es da überhaupt so intern ist.
Daten	Nutzung	Gleiche gilt für Nutzung(Ich muss noch mal hervorheben Wenn man da besonders darauf achtet, wer da wer sieht, welche Daten, wer darf die verändern? Ja, genau.)
Daten	Schnittstellen	Es muss immer auch definiert werden welche Konnektoren lasse ich zu.
Daten	Schnittstellen	Schnittstellen haben einen Zusammenhang zu Daten, weil ich gegebenenfalls einschränken kann.
Daten	Schnittstellen	Das Ganze Rollen rechte Konzept sollte sich auch auf die Schnittstellen übertragen. Also wenn ich jetzt als Anwender in der in der Low Code Plattform bin oder auch als Entwickler und ich darf nur bestimmte Schnittstellen anpacken und mit in meine Entwicklungslösung reinziehen
IT-Architektur	Geschäftskritikalität	Eine schlechte IT-Architektur kann im Zweifel dazu führen, dass die Definition der Single Source of Truth komplett kaputt geht. Dann ist das Vertrauen in die Daten nicht mehr hoch und der Nutzen der Anwendung geht ein bisschen verloren. Bei geschäftsunkritischen Fällen kann man vielleicht auch ein bisschen drüber hinwegsehen
IT-Architektur	Lebensdauer	Um das jetzt hier klarzumachen, würde ich einfach mal pauschal diese 0 rein nehmen, weil ich Architektur Entscheidungen dann nicht vom Lebensdauer abhängig machen würde. Ja.
IT-Architektur	Schnittstellen	Grundsätzlich sollten für die IT-Architektur immer Guidelines gemacht werden
IT-Sicherheit	Geschäftskritikalität	Sicherheit auf jeden Fall spielt das eine Rolle und bei kritisch und geschäftskritisch würde ich extra definieren.
IT-Sicherheit	Schnittstellen	Also wenn du die Anforderung hast, etwas zu verschlüsseln, dann muss das gemacht werden, ganz egal, ob es jetzt standardisiert, konfigurierbar, über Individualisierbar ist.
IT-Sicherheit	Schnittstellen	Potenziell kann man Schnittstellen auch gut überwachen, also dass man da überlegt. Gibt es irgendwelche Alarme, die man da rausziehen kann um Serviceanfragen Störungen? Untersuchen zu können oder auch direkt schon zu sehen, ist da ein Problem ist.
IT-Sicherheit	Schnittstellen	Bei individualisierten Schnittstellen muss man noch mal doppelt drauf schauen, weil da baut man ja neue Sachen, die noch nicht vorhanden sind.
Lösungserstellung	Anpassungsmöglichkeit	Würde ich sagen plus, weil dieser Prozess von Neuerungen, der muss mitgedacht werden, wenn es um Lösungserstellung geht, klar.
Lösungserstellung	Anpassungsmöglichkeit	Wenn man etwas eigenes entwickeln möchte, braucht es so eine Art Regelkatalog, um zu entscheiden, wann ich wirklich nutzerseitig etwas ausbaue,

Qualität	Geschäftskritikalität	Je höher die Auswirkungen von Fehlern sein kann, desto wichtiger ist auch, Qualität zu achten.
Qualität	Lebensdauer	Qualität niemals ganz vernachlässigen, aber wenn wir wussten, dass die Anwendung nicht lange im Betrieb ist, musste der Aufwand und der Nutzen im Verhältnis stehen. Ein ausführliches Testing dauert auch einfach und ist aufwendig. Dementsprechend waren die Qualitätsmaßnahmen auch bei Anwendungen, die nur kurz im Betrieb waren, nicht ganz so hoch.
Qualität	Lebensdauer	Qualität niemals ganz vernachlässigen, aber wenn wir wussten, dass die Anwendung nicht lange im Betrieb ist, musste der Aufwand und der Nutzen im Verhältnis stehen. Ein ausführliches Testing dauert auch einfach und ist aufwendig. Dementsprechend waren die Qualitätsmaßnahmen auch bei Anwendungen, die nur kurz im Betrieb waren, nicht ganz so hoch.
Qualität	Nutzung	Wenn es Abteilung ist, intern war, hat es ausgereicht für uns, wenn wir ein Feedback Loop mit einem Ansprechpartner aus der jeweiligen Abteilung gemacht haben.
Qualität	Nutzung	Während wir bei Abteilungsübergreifenden und Standortübergreifenden ganz ins besonders dann noch mal ganz eigene interne Qualitätsprozesse sozusagen drüber laufen lassen haben.
Qualität	Programmierfähigkeiten	Aber Familienkenntnisse hat ja oder nein, das spielt überhaupt keine Rolle, würde ich sagen. OK, es geht wirklich darum, dass es im Betrieb funktioniert,
Qualität		Es muss jetzt also, ich glaube, dass man eine Art Handbuch braucht, was sind übliche Dinge, die man zu prüfen hat, wenn man jetzt eine Lösung aufbaut und dann in den Betrieb überführt. Aber Familienkenntnisse hat ja oder nein, das spielt überhaupt keine Rolle, würde ich sagen. OK, es geht wirklich darum, dass es im Betrieb funktioniert und. Ich sag mal so, es gibt ja, es gibt ja schon auch diese Unit Test, die dann eher so aus der Lösung heraus schon durchgeführt werden. Aber das Wichtigste ist, dass es am Ende in der Anwendung der Lösung funktioniert, also quasi die die User Tests, die einen User Test und die müssen so oder so durchgeführt werden, deswegen würde ich das hier von allen 3 Fällen gleich behandeln okay alles klar.
Ressourcen	Anpassungsmöglichkeit	Die Anpassungen des Source-Codes sollte im Sinne des rollenrechte Konzepts so festgelegt werden, dass nur dedizierte Leute den Source Code verändern können
Ressourcen	Programmierfähigkeiten	Bei keinen bis wenig Programmierkenntnissen ist es relevant zu wissen, wieviel Erfahrung die jeweiligen Mitarbeitenden haben und je nachdem, wie viel Erfahrungen vorhanden ist, muss die Schulung anders aussehen.
Support	Anpassungsmöglichkeit	Die Art der Plattformanpassung hat einen Einfluss auf den Support. Es kommt darauf an, wie ich serviceanfragen Route, weil es entweder in der internen Verantwortung oder externen Verantwortung liegt, ja.
Support	Geschäftskritikalität	Also die Toleranz von Ausfallzeiten sinkt eben bei geschäftskritischen Themen, deswegen muss man da auch besonders drauf achten.

Support	Geschäftskritikalität	Auch unkritische Prozesse darf man nicht nur so liegen lassen, weil sonst, wenn man keinen Support anbietet, dann kann man es auch gleich nicht anbieten.
Support	Lebensdauer	Also die, die wenn ich schon von vornherein weiß, das wird etwas, was ich sehr, sehr lange im Betrieb habe, dann baue ich vielleicht auch dedizierte oder gehe noch mal intensiver in die Schulung meines Servicepersonals in der Serviceabteilung, wenn es mal ganz kurz ist.
Support	Nutzung	Dasselbe sehe ich jetzt für die Nutzung abteilungsintern (Auch unkritische Prozesse darf man nicht einfach nur so liegen lassen, weil sonst, wenn man keinen Support anbietet, dann kann man es auch gleich nicht anbieten.)
Support	Programmierfähigkeiten	Ja, hatten einen hatten Zusammenhang auch wieder, was das Routen angeht. Dass man da Expert*innen braucht, um das zu lösen.
Wissen	Anpassungsmöglichkeit	Die Lösungsbausteine müssen so dokumentiert sein, dass sie für alle Anwendenden nachvollziehbar sind.
Wissen	Nutzung	Standardübergreifend ist, würde ich vielleicht noch mal eine Sonderregel einführen
Wissen	Programmierfähigkeiten	Das muss halt auf einer anderen Art und Weise vielleicht auch bereitet sein, damit jemand mit oder ohne Entwicklungskenntnisse oder was kann etwas anfangen kann.

A.6 Expert*inneninterview Wirkbeziehungen I2

Regel	Merkmal	Zitat
Anforderungs-definition	Nutzung	Intern ist das ganz einfach, wenn man jeden Tag miteinander zusammenarbeitet und der gemeinsamen Sprache hat.
Anforderungs-definition	Nutzung	Spätestens ab abteilungsübergreifend, wird es dann intensiver.
Anforderungs-definition	Nutzung	Im Betrieb der Anwendungen nutzen wir Tool, wo der Fachbereich die Möglichkeit hat, in diesen Entwicklungs-Loop Ideen mit einzuspeisen.
Anforderungs-definition	Programmier-fähigkeiten	Die Citizen Developer bauen sich die Anwendung in einem No Code Tool. Die Einstiegshürde ist hier sehr gering.
Anspruchs-gruppen	Geschäftskriti-kalität	Bei unkritischen Anwendungen gehst du nach dem Standard vor, das heißt, du nimmst einfach das, was dir Organisation vorgegeben hat. Das Gleiche ist auch bei Abteilungsinterne. Bei kritischen oder geschäftskritischen Anwendungen muss ich ein eigenes Projektteam schaffen, ggfs. mit externen.
Anspruchs-gruppen	Nutzung	Bei Abteilungsübergreifenden Anwendungen muss ein eine Abteilung und bei Standortübergreifenden muss ein Standort die Führung übernehmen. Das heißt, ich muss ein eigenes Projektteam schaffen, ggfls. Mit externen
Anspruchs-gruppen		Ja, also grundsätzlich die wenig Erfahrung haben, die Arbeiten auch mit einem unserer Partner zusammen, das ist der Partner übernimmt die die Rolle der Projektleitung, holt sich diese Stakeholder rein und holt sich dort auch das Fachwissen ab. Von den einzelnen Abteilungen oder falls der Erwartungshaltung dies bringt einen Expert*innen mit rein, der das Fachwissen in diesem Meeting dann auch mit reinbringt.
Anspruchs-gruppen		Als erstes sollten die Verantwortlichkeiten geklärt werden und die sollten dann auch standardmäßig immer die gleichen sein
Anspruchs-gruppen		Im Endeffekt gibt es dann entweder prozessverantwortliche, die treffen sich oder organisieren auch diese Digitalisierung Initiativen im Unternehmen. Die holen sich entsprechend der Stakeholder rein und entwickeln erst mal den Prototypen und sorgen dann auch für die Umsetzung durch die IT und begleiten dann auch den gesamten Go Live Prozess mitsamt dem Change Requests, die an so einem Projekt auch dran hängt.
Betrieb	Geschäftskriti-kalität	Je kritischer der Prozess ist, desto häufiger müssen die Anwendungen angepasst werden.
Betrieb	Geschäftskriti-kalität	Die Plattform erledigt das für einen. Wir haben den gesamten Deployment Prozess mitsamt Versionierung so weit wie möglich automatisiert
Betrieb	Geschäftskriti-kalität	Du hast im Regelfall immer ein Entwicklungs-, ein Test- und ein Produktivsystem, um nichts kaputt zu machen
Daten	Geschäftskriti-kalität	Das Geschäftskritische geht vielmehr um die Sensibilität von Informationen. Da gibt es doch mal Sonderrunden, die man drehen beispielsweise Geschäftsgeheimnisse, wo du bis auf Datenbankebene runtergehst.
Daten	Nutzung	Wir arbeiten grundsätzlich nach dem Prinzip kein Zugang als Standard, also, du musst aktiv Zugriff geben, auf etwas, um zu sagen du siehst Daten

Daten	Nutzung	Wir arbeiten grundsätzlich nach dem Prinzip kein Zugang als Standard, also, du musst aktiv Zugriff geben, auf etwas, um zu sagen du siehst Daten, das lässt sich auch wieder je nach Unternehmensstruktur
Daten	Nutzung	Bei Standort übergreifend, ist da mit Sicherheit nochmal ein Thema, wenn du hast wenn du Datenquellen hast, wer das wird hier Datenquelle verwenden. Aus welchen Daten Quellen sehen, aus welchem Standort also beispielsweise darf Standort A die Lagerbestände vom Standort besehen?
Daten	Programmierfähigkeiten	und damit mit dem Skript auszuhelfen mit SQL, vor allem, wenn ich, wenn ich Daten Quellen, anzapfe aus anderen Systemen aber ist das, wenn es über SQL beispielsweise passiert, oft ganz sinnvoll sowas zu können
IT-Architektur	Nutzung	Die Frage ist auch, wie sich die Standorte in Bezug auf ihre IT-Infrastruktur unterscheiden. Wenn es Standort übergreifend ist, dann musst du dich nochmal intensiver damit auseinandersetzen, da ist natürlich noch mal eine Sonderrunde zu drehen pro Standort.
IT-Architektur	Schnittstellen	Eine lokale Plattform zur Automatisierung von Geschäftsprozessen lebt davon, dass sie optimal eingebunden ist.
IT-Sicherheit	Programmierfähigkeiten	Aber wir sagen, die Fachbereiche sind nie selber für Anwendungen verantwortlich, weil dann hast du eben wieder die ganzen Compliance Themen, wo ja auch Sicherheit usw. mit rein stützen
IT-Sicherheit		standardmäßig behandeln wir auch hier alle Anwendungen gleich. Wir haben einen sehr hohen Sicherheitsstandard, der automatisch mit drin steckt überall.
Lösungserstellung	Anpassungsmöglichkeit	Wenn Entwickler Code schreiben, dann haben wir dafür auch einen Standard wie der zu nutzen ist also, wenn wir nicht sehr hohen Standardisierungsgrad hier drinnen.
Lösungserstellung	Anpassungsmöglichkeit	Wir müssen bestimmte wir eben wie gesagt an hohen Standardisierungsgrad reinbringen, die ja sehr hohe Sicherheit auch mit sich bringt und dann lassen wir keinen Grenzen zu.
Lösungserstellung	Nutzung	Eben beim du damit also mit so einer Plattform magst du Dutzende hunderte Anwendungen und tun, du musst den Standard setzen, sonst werden die Mitarbeiter irgendwann nicht mehr mitspielen.
Lösungserstellung	Programmierfähigkeiten	Der Prototyp der Fachbereiche wird zum fertigen Entwickeln an die IT übergeben
Lösungserstellung	Programmierfähigkeiten	Wir lassen die Anwendung durch die Fachbereiche als Prototyp vorbauen und erlauben den Fachbereichen so genau zu definieren, wie der Prozess aussieht und wie die Anwendung auszusehen hat.
Qualität	Geschäftskritikalität	Bei Low-Code-Projekten muss das Testen bei kritischen und geschäftskritischen Prozessen mehr beachtet werden, damit auch alles glatt läuft
Ressourcen	Anpassungsmöglichkeit	Ja, und es gibt also es gibt grundsätzlich Schulungen. Es gibt dann auch mal Spezialschulungen eben für Systeme, Administratoren für SPK Entwickler da gibt es nur ein anderes andere Schulungen.
Support		Das heißt, du hast immer den gleichen Standard, was dann auch bei Service Anfragen und Störungen den Wartungsaufwand minimiert

Wissen	Schnittstellen	Aber was wir nicht haben, also wir haben keinen Market Place für Schnittstellen, wir gehen hier ja pragmatisch vor im Endeffekt, wir haben eine ne User Community die sich austauscht, die auch dann ihre Projekte auf Gitarre usw. teilt.
Wissen		Wissen ist standardisiert rund um diese Plattform.
	Schnittstellen	Standard Schnittstelle über Theobald, das Macht glaub ich jeder so zu SAP hin und alles andere fangen wir über APIs ab, wo wir mit Rest und Soap Standort arbeiten.

A.7 Expert*inneninterview Wirkbeziehungen I3

Regel	Merkmal	Zitat
Anforderungs-definition	Anpassungs-möglichkeit	Wir sagen gleich vorab, ob sich der Use Case eignet mit Low-Code umzusetzen.
Anforderungs-definition	Anpassungs-möglichkeit	Wir sagen gleich vorab, ob sich der Use Case eignet mit Low-Code umzusetzen.
Anforderungs-definition	Nutzung	Richtung Anforderungsdefinition ist es wichtig, frühestmöglich all beteiligten an einen Tisch zu kriegen, um langen Integration Zyklen zu vermeiden.
Anforderungs-definition	Programmier-fähigkeiten	Bei der Anforderungsdefinition, ist es wichtig zu verstehen was eine Anforderung, für Auswirkungen mit sich bringt, zum Beispiel muss man verstehen, dass Offline Fähigkeit bei Apple Geräten immer was sehr spezielles ist.
Anforderungs-definition	Programmier-fähigkeiten	Wir sagen gleich vorab, ob sich der Use Case eignet dieser Use Case für Citizen Developer empfiehlt
Anspruchs-gruppen	Geschäftskriti-kalität	Ein ideales Team besteht aus mehreren Stakeholdern je nach Komplexität Grad der Anwendung
Anspruchs-gruppen		je nachdem, wie komplex die Anforderungen ist oder beziehungsweise die Applikation, die realisiert werden muss, brauchst du entweder ein größeres Team oder kleineres und das ist halt das das Phänomen,
Daten	Geschäftskriti-kalität	Das Thema Geschäftskritikalität ist häufig mit den integrierten Daten verbundene, weil sich die Daten auf die Anwendungen beziehen
Daten	Schnittstellen	In Verwendung und dann hat man meistens unterschiedliche Personen, die die Schnittstellen administrativ betreuen und die haben dann natürlich intern dann das sagen darüber, ob die Daten freigegeben werden
Daten	Schnittstellen	Meistens müssen die Schnittstellen in den anderen Systemen freigegeben werden, auf die zugegriffen werden kann
Daten	Schnittstellen	Genau oder beziehungsweise die Berechtigung zur Schnittstellenerstellung internem Unternehmen haben, weil das wäre meistens eine strategische Entscheidung.
Daten		Und ganz häufig ist es so, dass du hier als Citizen Developer gar nicht viel machen kannst, weil für die anderen Systeme die Berechtigungen fehlen
IT-Sicherheit	Schnittstellen	Die Erstellung der individualisierten Schnittstellen hängen auch ganz von den Systemen und deren Sicherheitsanforderungen ab.
Lösungserstel-lung	Anpassungs-möglichkeit	Wenn man nativ entwickelt ist es ein Riesenaufwand, wenn man ein einheitliches Design pflegen muss, dann muss man das ja auch alles definieren an irgendwelchen Stellen, dass jeder einheitlich ist und so und das ist halt bei Silvia vorgegeben
Lösungserstel-lung	Programmier-fähigkeiten	Erfahrene Programmierer programmieren vieles nativ und nutzen kaum die vorgefertigten Module
Ressourcen	Programmier-fähigkeiten	Wir haben intern gemerkt, dass sich erfahrene Programmierer sich viel zu stark ihre Programmierfähigkeiten verlassen und die Vorteile von Low-Code teilweise gar nicht nutzen
Ressourcen	Schnittstellen	Beim Erstellen neuer Schnittstellen muss man teilweise sehr stark individualisieren und dann musst du in SAP eintauchen und SAP programmieren können, um die Schnittstelle bereitzustellen

Support	Geschäftskritikalität	Die kritischeren Apps werden im Support System höher priorisiert.
Support	Programmierfähigkeiten	Mehrere Möglichkeiten des Supports, zuerst beginnt man mit einem Forum und der Dokumentation, dann kommen die Support Sessions. Wo zuerst versucht wird, das Problem mit erfahrenen interne Entwicklern zu lösen bevor Sie eine externe Support Session buchen. In den Support Sessions sitzen von Citizen Developer bis zu IT Professionals und es hängt sehr von der Anwendung ab.
Wissen	Nutzung	Wir haben auch ne eine Matrix und da sie beziehen wir halt noch die Stakeholder mit ein und sagen Personen ist XY ist verantwortlich für XY Thema

A.8 Expert*inneninterview Wirkbeziehungen I4

Regel	Merkmal	Zitat
Anforderungs-definition	Geschäftskriti-kalität	Bei kritischen und geschäftskritischen Anwendungen sollte nochmal besser geprüft werden, ob das wirklich die richtigen Anforderungen sind und ob man sie auch wirklich umgesetzten kann.
Anforderungs-definition	Programmier-fähigkeiten	Mit mehr Erfahrungen kann man die Anforderungen auch besser einschätzen
Anforderungs-definition	Schnittstellen	Bei der Anforderungsdefinition beurteilen wir inwiefern das mit standardisierten Schnittstellen umgesetzt werden kann.
Anspruchs-gruppen	Programmier-fähigkeiten	Also wir haben solche Communities, da würde ich aber sagen, dass Leute mit keiner Erfahrung und mit wenig Erfahrung zusammen sind, die IT-Spezialisten sind da weniger vertreten
Anspruchs-gruppen	Programmier-fähigkeiten	Der gegenseitige Austausch ist auch wertvoll für die, die Erfahrung haben, um sich einfach so die neuesten Erkenntnisse gegenseitig präsentieren zu können, Anwendungen zur Inspiration zu erhalten und sich gegenseitig bei Fragestellungen weiterzuhelfen
Betrieb	Geschäftskriti-kalität	Je kritischer der Prozess ist, das so detailliert muss der Bereitstellungsprozess definiert werden
Betrieb	Nutzung	Je mehr Abteilungen die Anwendung nutzen, umso komplexer wird der Prozess meine Anwendung einzuführen.
Betrieb	Schnittstellen	Je mehr individualisierte Schnittstellen ich habe, desto komplexer wird der Betrieb und Support.
Daten	Geschäftskriti-kalität	Je geschäftskritischer die Anwendung ist, desto wichtiger ist ein Zugriffsmanagement
Daten	Nutzung	Wenn ich Standort übergreifend etwas Nutze, muss mein Zugriffsmanagement nochmal detaillierter verwaltet werden
Daten	Schnittstellen	Standard Schnittstellen können wir ohne Probleme nutzen, alles weitere muss beantragt werden.
IT-Architektur	Nutzung	Bei Standortübergreifender IT-Architektur muss nochmal mehr abgestimmt werden.
IT-Architektur	Schnittstellen	Power BI kann halt nicht im Standard auf SAP zugreifen, dann braucht man sich entweder Round, wo die SAP Daten in der andere SQL Datenbank übertragen werden, so dass wir mit VW ja wieder darauf zugreifen können. Natürlich wäre es schön, wenn wir diese konfigurierte Schnittstelle zu SAP direkt hätten.
IT-Architektur	Schnittstellen	Zu Beginn wurde einmalig ein Standard festgelegt, den wir so nutzen können und auf den wir freigeschaltet sind und wissen, was wir da nutzen können.
IT-Architektur	Schnittstellen	Zu Beginn wurde einmalig ein Standard festgelegt, den wir so nutzen können und auf den wir freigeschaltet sind und wissen, was wir da nutzen können.
IT-Sicherheit	Nutzung	Die Standorte haben vielleicht unterschiedliche Richtlinien in Richtung Sicherheitspraktiken. Das ist zumindest im EU Ausland und in der EU unterschiedliche und Deutschland und deutsches Ausland auch.
IT-Sicherheit	Schnittstellen	Je individualisierter die Schnittstelle ist, umso höher muss auch die Sicherheit mit einbezogen werden.
Lösungserstellung	Nutzung	Bei der Entwicklung des Standards für unseren Standort wurden mehrere Abteilungen miteinbezogen. Für einen

		Standortübergreifenden Standard würden wir anders vorgehen.
Qualität	Geschäftskritikalität	Je kritischer ein Prozess wird, desto mehr guckt dann auch die IT drauf.
Qualität	Nutzung	Vor dem Deploy wird die Anwendung im Entwicklerteam überprüft und teilweise durch die Expert*innen im Fachbereich überprüft.
Qualität	Nutzung	Gleiches gilt für Nutzung (Je kritischer ein Prozess wird, desto mehr guckt dann auch die IT drauf.)
Ressourcen	Programmierfähigkeiten	Wir haben viele Freiräume, uns das Wissen selbst anzueignen, z. B. über LinkedIn Learning oder YouTube Videos. Dazu benötigen wir dann Zeit, um sich das selbst beizubringen.
Support	Geschäftskritikalität	Das gleiche kannst du bei Support übernehmen (Je kritischer der Prozess ist, das so detailliert muss der Bereitstellungsprozess definiert werden)
Support	Programmierfähigkeiten	Mit mehr Erfahrungen kann jemand besseren Support leisten.
Wissen	Geschäftskritikalität	Je kritischer ein Prozess wird, desto mehr muss ja dokumentiert werden.
Wissen	Lebensdauer	Bei uns ist es so, dass häufig auch Studenten Dinge entwickeln und da hat das Problem, wenn die weg sind wer macht das weiter
Wissen	Schnittstellen	Bei einer standardisierten Schnittstelle muss zwar dokumentiert werden, aber je individueller es wird, desto höher muss da auch der Dokumentation Aufwand sein
Wissen	Schnittstellen	Bei einer standardisierten Schnittstelle muss zwar dokumentiert werden, aber je individueller es wird, desto höher muss da auch der Dokumentation Aufwand sein

A.9 Expert*inneninterview Wirkbeziehungen I5

Regel	Merkmal	Zitat
Anforderungs-definition	Nutzung	And then it also of course quality also boils down to requirements management because if you have a requirements management that goes like this right to left to right, left to right and there's not a really good vision behind the product. This is also something you can find back inequality of the application.
Anspruchs-gruppen	Nutzung	So let's say if you develop something cross locational, of course your project team in in total for developing this is has a different structure or different people inside than if you do something department specific. If you're type one, then probably the one doing the clicking is also the one who's the product owner.
Anspruchs-gruppen	Programmier-fähigkeiten	But sometimes you need like something really specific and we have these projects as well where normally an external design agency just designs something and we have to implement that and so we also build customer facing applications. So you need also specific skills for that.
Betrieb	Anpassungs-möglichkeit	So yeah, but on the other hand of course if you do upgrades and if you want to do changes there, yeah you're looking into a custom made code.
Betrieb	Lebensdauer	And if you're not that worried about maintenance because you don't have any maintenance because you're only going to use it a year, then you might invest a little bit less than that.
Betrieb	Schnittstellen	So maybe this custom made code uses an API of the platform that's deprecated in a version and then yeah so yeah you do create some kind of overhead.
Daten	Programmier-fähigkeiten	If you have less experiences software developers, then you would need to prepare all these data sources for your for your business developers, because they don't have the skills or knowledge on what data they can use. So the first thing is only read
Daten	Schnittstellen	For writing, I would have to create all my data validation rules and stuff like that.
Daten	Schnittstellen	The first thing is reading data. So you'd really have to think about OK, what data can they access and is that is that sensitive information. And I would have to make an API really simple, one standardize that, that the business user can use it. Yeah, for writing, yeah, I would have would have to create all my data validation rules and stuff like that.
Daten	Schnittstellen	And you, if you look at Power Apps, you know everything that's in the Microsoft ecosystem, that works pretty good because they have data access rights and that's all in one system, right? That works pretty good. So if you want to have a Power Apps app and you want to show my emails, that's really easy in Power Apps, show my emails. But then, oh, we have a shared mailbox and all the mails that come into this mailbox need to be processed and they need to be moved to another. Yeah. Then you run into issues because then you need to do custom stuff and so especially for integrations, you run into this, these limitations quickly.

Daten		But as soon as you're building applications that's a two-way street, right? Because those are probably some data must go back and then you run into a whole lot more issues. So, then you need to also, well, either forbid more or control more.
Daten		You know what you call citizen development ship right? Business building their own software. This works pretty good for Power BI, for BI tools because then you have a one-way Street. It's easy, we can control what data you get, and you create all the visuals you want for this and all the down drills. And if you want to do this way or that way you know what you want, build it.
Lösungserstellung	Anpassungsmöglichkeit	So what we what we've created for instance is a is a is a design system and this this has like a broad set of standard UI components that can be reused by these local developers and also by product owners because they have these example pages with all the different UI elements you can pick in the standard, right?
Lösungserstellung	Nutzung	I would say this is also the same for the usage the how many like if it's only for a small department it's rather less complex. So, you could probably use some more standards, Yeah, yeah, yeah, sure.
Lösungserstellung	Programmierfähigkeiten	So, we have modelling guidelines that say OK, how do you make your application and how do you do naming conventions and stuff like that, best practices. So, we train everyone in that and then we have peer reviews that we check that that we also follow these rules.
Lösungserstellung		I think it's important to understand or to note that there are different levels of components. So, for instance, A component can be like a complete collection of data structures, logic and UI. Like in one, right? This is a module, let's call it a planning module, which has a lot of stuff in it. But then in low code you also have like a component that's maybe like a button. A button is also a component, but a button in in low codes. It could be that there are some requirements which you cannot meet with the standard button in your local platform. And then you would need to build an extension in high code for that specific button, right? That's also a component to this button. So, the, the, the, the, the, the level on which this component lives can be a pretty big one, but there can also be a really small one.
Qualität	Geschäftskritikalität	for business critical you do more testing and do automated testing and stuff like that. So, the more business critical there is the bigger the, the team is that's going to analyses what's the impact on well yeah, what's the, what's the impact of a failure.
Qualität	Geschäftskritikalität	And then for the more critical your application is of course the more additional quality measures are added, but that's something that's discussed with our customer and then it's custom right. It depends on also effort budget risk it's a risk budget discussion and what would be some like additional measures testing and testing of course there's they're always testing but if you're like a non-critical application then you wouldn't even do tests right. So, then you don't do really testing and then for critical you do some testing and for business critical you do more testing and maybe do automated testing and stuff like that. So, the more business

		critical there is the bigger the, the team is that's going to analyses what's the impact on well yeah, what's the, what's the impact of a failure.
Qualität	Lebensdauer	But if you would build for short term, you would worry less about quality.
Qualität	Nutzung	Because the developer is the user most probably or is close to the users. So, the impact of a bug is really low and especially with logo we can fix it right away and then you know your colleague on the other side of the table says hey why doesn't this work. So, the more business critical there is the bigger the, the team is that's going to analyses what's the impact on well yeah, what's the, what's the impact of a failure.
Support	Geschäftskritikalität	And then for more, more critical applications, you would have dedicated support that that helps you with that was who, So we have. So, for instance, we have a DevOps team that's dedicated 3 people and our customers can have a contract with us so that they can call anytime with issues. And you could also have this internally in your in your organization, of course, if you have like a big center of excellence, yeah, support is a is a big part of that. But yeah, if you have like these low, low-level applications, yeah, you might, you might help them with questions.
Support	Nutzung	And then for more, more critical applications, you would have dedicated support that that helps you with that was who, So we have. So, for instance, we have a DevOps team that's dedicated 3 people and our customers can have a contract with us so that they can call anytime with issues. And you could also have this internally in your in your organization, of course, if you have like a big center of excellence, yeah, support is a is a big part of that. But yeah, if you have like these low, low-level applications, yeah, you might, you might help them with questions.
Support	Nutzung	Simple applications I would say support and management is in in the department that created the IT right It's their issue, their software, they know how it works, let them fix it.
Wissen	Anpassungsmöglichkeit	Then you need coding skills and does that somehow those like adding new components with high code affect the quality of your software or the, I don't know, knowledge management, anything like looking into the left side it could because of course you need different skills, so different people, so you have more knowledge transfer. On the other hand, the way these platforms or at least I of course know most about Mendi is they abstract these extensions in a way that as soon as you've made the extension, So for instance this new button with a new feature, right, You can make that as a new low code component. So, the next user who uses that button doesn't know or doesn't need to know that it's that it's pre made in in in in high code or that it's a custom.
	Programmierfähigkeiten	There's a there's a difference between I can write code, or I can, you know, make software. And that means that the skills of the developer have really changed. Even though it's a professional software developer, it's you don't need to have these coding skills and that's a that's a big, big difference.

A.10 Expert*inneninterview Wirkbeziehungen I6

Regel	Merkmal	Zitat
Anforderungs-definition		Die würden aber hoffentlich irgendwie auffallen, der stabilisiert nicht in die Richtung, aber der eigentliche Hirn Schmalz steckt in der Regel in dem, was du da mit dem Business wert wofür machst du das? Technik folgt den Anforderungen sowas Bernd eben auch sagt also, wenn du so 10000 Zeilen Code und ganz viele Sachen noch mit löst ne, das willst du nicht?
Anforderungs-definition		Wofür machten das, was ihm wichtig und das zieht sich ja letztlich runter bis in den Betrieb, wenn ich weiß, da geht jeden Abend um 5 nach Hause, dann brauche ich aber ganz andere Gedanken machen und du kannst tatsächlich Kohle sparen, indem du deine Systeme nicht riesig skaliert oder vielleicht nachts ausstellst, weil dann bist du Grüne solche Gedanken kannst du auch sowas mitnehmen, wenn du die richtigen Leute an den Tisch setzt.
Betrieb	Nutzung	Aber eigentlich also, wenn du, wenn du sowas wie eine digitale Souveränität anstrebst, möchtest du eigentlich in Haus und Software Team haben. Wie auch immer das gestrickt ist, das in der Lage ist, diesen Betrieb zu sicherzustellen. Am besten das Team, das auch dieses Produkt baut.
IT-Architektur		Je nach Business möchtest du heute vielleicht mal bisschen mehr Kapazität haben, oder? Gewisse Projekte sind vor Weihnachten kritischer als mit einem Jahr. Also möchtest du solche Dinge vielleicht unter deiner Kontrolle haben? Am das musst du aber auch genau so entscheidend also eine Infrastruktur aufsetzen, die, dass die das kann.
IT-Sicherheit	Nutzung	Das kann aber ein kleines Team im kleinen Rahmen auch selber aufbauen, also das ist nicht so groß. Dann Datensicherheit ist natürlich auch irgendwo immer ein bisschen was passiert, wenn jemand von außen einbricht, wenn das nur Software ist, die interne läuft, kann man das durchaus kleiner und einfacher handhaben.
IT-Sicherheit	Programmier-fähigkeiten	Da braucht man dann eine kleine PKI, also private Key Infrastrukturen. Das war etablieren muss ja, das kann man halbwegs gut auslagern.
IT-Sicherheit	Schnittstellen	Also stehst du schon sicher das Ende, die API nach außen anbietest, dass Du die kappen kannst, zum Beispiel irgendwelche API Gateway oder ähnliche Mechanismen in der Praktikant beim Kunden quasi irgendwas automatisiert und vorher jetzt 3000000 Nachrichten pro Sekunde auf die API Distant System nicht runter geht, also schützt das schon das sind aber so Standard.
Qualität		Das ist aber in den Prozessen bisschen integriert, also wir haben kross-funktionale Teams, wir haben in der Regel ein 4 Augen Prinzip Wir haben Code Reviews also auch mein Code oder der vom Bernd geht nicht ohne Rewe und die Produktion. Wir haben automatisierte Tests versuchen also bin nicht, behaupten wir machen Test Driven Development, aber wir haben in der Regel eine Absicherung auch der fachlichen
Wissen	Lebensdauer	Wir haben jetzt in der Iteration gelernt deine Arbeitsweise ist gar nicht wie angenommen und dann muss ich den

		Code anfassen, ansonsten habe ich was falsch gemacht. Und diese auch Kosten und diese Veränderung zu tracken und das Lernen dafür ist dieses zusammen da, da kann die Doku helfen.
Wissen	Lebensdauer	Hm, es gibt so Konzepte wie Proof of Concept oder Prototypen Dinge, die du wegschmeißt, wenn sie oder ablöst, wenn sie sich, wenn sie gezeigt haben, dass sie funktioniert.
Wissen	Lebensdauer	Also ist immer eine gute Idee tatsächlich in. Der Punkt ist halt an der Stelle Mensch ist halt immer sehr. Dann ist die App. Was sollen wir jetzt dokumentieren, was nicht? Was lohnt sich?
Wissen	Nutzung	Also integriert in den Build Prozess mit dem Artefakt des in die Produktion deployed wird, wird gleichzeitig eine Dokumentation mit woandershin, Diploid oder hingelegt und du hast quasi im in deinem Versionskontrolle in Tracking auf dieses Ticket was du gelöst hast auf die Story wofür hast du das gemacht?
Wissen		Das Code und die Modelle, die du beschreibst, sind räsoniert der Code, die beschreiben, wie deine Anwendung aussehen und du kannst, wenn du es richtig gut machst, auch mit Tests und ähnlichem. Unterstützen also deine Dokumentation wird aus dem Kot oder aus einem Stück Kot Textbeschreibung im Repo erzeugt und auch nur, wenn der Test durchläuft.
Wissen		Ist in der Regel ja auch eine Integration, in der größere Anwendungslandschaft es ist ja quasi eine Anwendung, die sich das ist, ja auch diese loco Sachen ich also zumindest die ganz links du machst dir mal schnell den Bericht und dann guckst du, ob das was wird oder das bleibt oder kriegst du quasi zahlen oder so zusammen? Das ist aber integriert in eine größere Anwendung und den Kontext kannst du schon mit und die Erwartungen mit dokumentieren. Auf für Baukosten des Was willst du bezwecken mit dem Ding und wie Greif des technisch darein frisst es einfach Nachrichten hat, das Zugriff auf alles oder nur ein bisschen und dann ja

A.11 Expert*inneninterview Wirkbeziehungen I7

Regel	Merkmal	Zitat
Anspruchs-gruppen	Programmier-fähigkeiten	Also vor Corona war so viel mehr, dass wir sehr die Teams vor Ort zusammensitzen und dann auch vor Ort in einem Raum sitzen oder wenn auch auf einer Fläche und da viel kommunizieren oder einfach auch die erfahrenen n bisschen bei den weniger erfahren und bisschen so auf die Finger schauen, hängen wir jetzt da haben sich irgendwie geistig verbrannt, oder?
Daten		Was auch wichtig ist, ist immer zu wissen, ob es personenbezogene Daten dabei sind.
IT-Sicherheit	Geschäftskritikalität	Da haben wir jetzt auch Kunden, die da wirklich zwischen internen und externen Anwendungen unterscheiden, die externen Anwendungen zum Beispiel werden vom weiteren dritten Sicherheitsfirma Penn getestet, also die gucken wirklich, ob du da jetzt irgendwas aushebeln können
IT-Sicherheit	Schnittstellen	Also im Sinne von natürlich, dass die Daten manipulierbar und nicht einsehbar übertragen werden, ist ein wichtiger Aspekt.
IT-Sicherheit	Schnittstellen	Was das macht man auch in Oberflächen, also Circuit Breaker also Sicherung die werden zu viel Last drauf, ist einfach durch kühlen also wirklich eine Software durchführen und dann mal für eine Minute einfach der Dienst einstellen, explizit, dass die Gegenseite merkt, sie macht doch irgendwas Mist. [...] Ja, genau, aber das ist oft schon in den Plattformen mit drin. Ich könnte auch vorstellen, dass, wenn jemand der Logo Plattform anbietet und dann sagt, ihr könnt ihr jetzt zum Beispiel so ein Dienst oder so Oberfläche nach außen geben, dass das dann schon unten drin mit eingebaut ist.
Lösungserstellung	Programmier-fähigkeiten	Auf jeden Fall also die, die die Flug Höhe der Lösung muss natürlich zum Problem und zum technischen Unterbau immer passen. Und wenn du dann siehst, dass der 3 Tage nur Hacker und Tausende von Zeilen Code schreibt und eigentlich ist es irgendwie ein ganz einfaches Problem, oder ein Sohn dranhängst und noch irgendwie ein Config schreibst. Dann sagt mir hier auch mal kurz meistens mal weg das geht anders einfacher also die die Komplexität der Lösung muss zu technischen Umfeld und zum Problem immer passend und das ist auch für Anfänger und ganz kritisches Ding.
Wissen	Geschäftskritikalität	Egal also es gibt ja Software, die jetzt irgendwie Sicherheit relevant ist, was zum Teil ja auch für Maschinen betrieb oder sowas ist, das ist da, wo der Maschine und Menschen zusammen auf irgendeine Fläche agieren oder ich hab das für Flugsicherung Software gebaut, da ist natürlich auch immer das Interesse da, das Software dann vernünftig reagiert.
Wissen	Schnittstellen	In Schnittstellen zwischen den System, da brauchst du auch eine formale Definition was geht rein und was geht raus?

A.12 Zuordnung von Maßnahmen zu normalen Wirkbeziehungen

Regel	Merkmal	Intensität	Maßnahmennummer
Anforderungsdefinition	Anpassungsmöglichkeit	+	O6
Anforderungsdefinition	Programmierfähigkeiten	+	O7
Anforderungsdefinition	Schnittstellen	+	O8
Anforderungsdefinition	Schnittstellen	+	O8
Anspruchsgruppen	Programmierfähigkeiten	+	O2
Betrieb	Geschäftskritikalität	+	O12
Betrieb	Geschäftskritikalität	+	T4
Betrieb	Lebensdauer	+	O13
Betrieb	Nutzung	+	O14
Daten	Geschäftskritikalität	+	O16
Daten	Nutzung	+	O16
Daten	Programmierfähigkeiten	+	O18
Daten	Schnittstellen	+	O17
Daten	Schnittstellen	+	T6
IT-Architektur	Schnittstellen	+	O3
IT-Architektur	Schnittstellen	+	O4
IT-Architektur	Schnittstellen	+	T1
IT-Architektur	Schnittstellen	+	T2
IT-Sicherheit	Programmierfähigkeiten	+	O23
IT-Sicherheit	Programmierfähigkeiten	+	O24
IT-Sicherheit	Schnittstellen	+	T10
IT-Sicherheit	Schnittstellen	+	T8
Lösungserstellung	Anpassungsmöglichkeit	+	O10
Lösungserstellung	Anpassungsmöglichkeit	+	T3
Lösungserstellung	Nutzung	+	O9
Lösungserstellung	Programmierfähigkeiten	+	O10
Lösungserstellung	Programmierfähigkeiten	+	O11
Qualität	Geschäftskritikalität	+	O5
Qualität	Lebensdauer	+	O5
Qualität	Nutzung	+	O5
Ressourcen	Programmierfähigkeiten	+	O1
Support	Anpassungsmöglichkeit	+	O15
Support	Geschäftskritikalität	+	O15
Support	Nutzung	+	O15
Support	Programmierfähigkeiten	+	T5
Wissen	Lebensdauer	+	O20
Wissen	Nutzung	+	O19
Wissen	Nutzung	+	T7
Wissen	Schnittstellen	+	O21

A.13 Zuordnung von Maßnahmen zu starken Wirkbeziehungen

Regel	Merkmal	Ausprägung	Intensität	Maßnahmennummer
Anforderungsdefinition	Anpassungsmöglichkeit	Low-Code	++	O26
Anforderungsdefinition	Geschäftskritikalität	Kritisch	++	O50
Anforderungsdefinition	Geschäftskritikalität	Geschäftskritisch	++	O50
Anforderungsdefinition	Nutzung	Abteilungsübergreifend	++	O50
Anforderungsdefinition	Nutzung	Standortübergreifend	++	O50
Anforderungsdefinition	Programmierfähigkeiten	Keine	++	O26
Anforderungsdefinition	Programmierfähigkeiten	Keine	++	O26
Anforderungsdefinition	Programmierfähigkeiten	Wenig	++	O26
Anforderungsdefinition	Programmierfähigkeiten	Wenig	++	O26
Anspruchsgruppen	Geschäftskritikalität	Kritisch	++	O27
Anspruchsgruppen	Geschäftskritikalität	Geschäftskritisch	++	O27
Anspruchsgruppen	Nutzung	Abteilungsübergreifend	++	O27
Anspruchsgruppen	Nutzung	Standortübergreifend	++	O27
Anspruchsgruppen	Programmierfähigkeiten	Keine	++	O25
Anspruchsgruppen	Programmierfähigkeiten	Wenig	++	O25
Betrieb	Anpassungsmöglichkeit	Source Code	++	O39
Betrieb	Geschäftskritikalität	Geschäftskritisch	++	O38
Betrieb	Nutzung	Standortübergreifend	++	O38
Betrieb	Schnittstellen	Individualisiert	++	O40
Daten	Geschäftskritikalität	Geschäftskritisch	++	O43
Daten	Nutzung	Standortübergreifend	++	O43
Daten	Schnittstellen	Individualisiert	++	O44
IT-Architektur	Geschäftskritikalität	Kritisch	++	O28
IT-Architektur	Geschäftskritikalität	Geschäftskritisch	++	O28
IT-Architektur	Nutzung	Standortübergreifend	++	O35
IT-Architektur	Schnittstellen	Konfigurierbar	++	O29
IT-Sicherheit	Geschäftskritikalität	Geschäftskritisch	++	O49
IT-Sicherheit	Nutzung	Standortübergreifend	++	O49
IT-Sicherheit	Schnittstellen	Individualisiert	++	O48
Lösungserstellung	Anpassungsmöglichkeit	Source Code	++	O37
Qualität	Geschäftskritikalität	Kritisch	++	O30
Qualität	Geschäftskritikalität	Geschäftskritisch	++	T9
Qualität	Geschäftskritikalität	Geschäftskritisch	++	O30
Qualität	Lebensdauer	Mehrere Jahre	++	O30
Qualität	Lebensdauer	Lang	++	O30
Qualität	Nutzung	Standortübergreifend	++	O36
Ressourcen	Anpassungsmöglichkeit	Source Code	++	O32
Ressourcen	Anpassungsmöglichkeit	Source Code	++	O33
Ressourcen	Schnittstellen	Individualisiert	++	O34
Support	Geschäftskritikalität	Kritisch	++	O31
Support	Geschäftskritikalität	Geschäftskritisch	++	O41
Support	Geschäftskritikalität	Geschäftskritisch	++	O31
Support	Lebensdauer	Viele Jahre	++	O42

Support	Nutzung	Abteilungsübergreifend	++	O31
Support	Nutzung	Standortübergreifend	++	O31
Wissen	Anpassungsmöglichkeit	Source Code	++	O47
Wissen	Geschäftskritikalität	Geschäftskritisch	++	O46
Wissen	Nutzung	Standortübergreifend	++	O45

A.14 Vollständige Liste der Maßnahmen

Maßnahmen-nummer	Maßnahme
O1	Low-Code-Schulungen abhängig von Programmierkenntnissen und Komplexität der geplanten Anwendungen erarbeiten
O2	Center of Excellence zum Austausch und zur kontinuierlichen Verbesserung einführen
O3	Guidelines für IT-Architektur definieren
O4	Zu Beginn standardisierte Schnittstellen festlegen
O5	Feedbackschleife mit Anwender*innen vor Inbetriebnahme der Anwendung einführen
O6	Kriterienkatalog zur Überprüfung der Low-Code-Eignung erstellen
O7	Richtlinien zur Überprüfung der notwendigen Programmierkenntnisse zur Anwendungsentwicklung einführen
O8	Übersicht über integrierbare Daten erstellen, um Datenbedarf der Anwendung abgleichen zu können
O9	Standards unter Berücksichtigung aller Nutzer*innen einführen
O10	Modellierungsrichtlinien definieren
O11	Lösungserstellung durch erfahrene Low-Code-Entwickler*innen überprüfen lassen
O12	Kommunikation für die Bereitstellung neuer Anwendungen festlegen
O13	Aufwand für die adaptive Wartung abhängig von der Lebensdauer der Anwendung festlegen
O14	Verantwortlichkeiten für den Betrieb der Anwendungen festlegen
O15	Supportprozesse mit Entwickler*innen der Anwendung definieren
O16	Allgemeine Regeln zum Datenumgang und -zugriff definieren
O17	Verantwortung und Zugriffsrechte für Schnittstellen definieren
O18	Technische Möglichkeiten des Datenzugriffs aus anderen Systemen identifizieren
O19	Verantwortlichkeiten für Anwendungen festlegen
O20	Umfang der Dokumentation abhängig von der Lebensdauer der Anwendung festlegen
O21	Schnittstellen dokumentieren
O22	Richtlinien zur Dokumentation und Verständlichkeit erstellen
O23	Betriebssicherheit sicherstellen
O24	Notwendigkeit der externen Unterstützung prüfen
O25	Austausch zwischen unerfahrenen Entwickler*innen durch regelmäßige Meetings gezielt fördern
O26	Anforderungsdefinition und technische Machbarkeit durch erfahrene Low-Code-Entwickler*innen überprüfen lassen
O27	Verantwortlichkeiten für die Koordination der Projektteams festlegen und ggfs. externe einbinden
O28	Single Source of Truth für alle in die Low-Code-Plattform integrierten Daten definieren
O29	Implementierung von konfigurierbaren Schnittstellen für nicht vorhandene Standard-Schnittstellen prüfen
O30	Qualitätsprüfung durch IT-Spezialist*innen vor Inbetriebnahme der Anwendung einführen
O31	Mehrstufiges Supportsystem aufbauen (Dokumentation und Forum, interne und externe Ansprechpartner*innen)
O32	Im Rollen und Rechtemodell festlegen, wer den Source Code der Plattform ändern darf
O33	Zusätzliche Schulungen für Source Code Anpassungen durchführen
O34	Zusätzliche Schulungen für individualisierte Schnittstellen durchführen
O35	Individuelle IT-Architektur je Standort berücksichtigen und abstimmen
O36	Qualitätsprüfung durch weitere Nutzer*innen vor Inbetriebnahme der Anwendung einführen
O37	Richtlinien zur Überprüfung der Notwendigkeit einer nutzerseitigen Source Code Anpassung einführen
O38	Prozess zur Ablösung der alten geschäftskritischen Anwendung definieren

O39	Bei Deployment oder Versionsupdate die individuellen Code Bausteine überprüfen
O40	Bei Deployment oder Versionsupdate die Konfiguration der individuellen Schnittstellen überprüfen
O41	Service Level Agreements festlegen
O42	Intensivere Schulung des Servicepersonals durchführen
O43	Anwendungsspezifische Regeln zum Datenumgang und -zugriff definieren
O44	Schnittstellenspezifische Verantwortung und Zugriffsrechte für individuelle Schnittstellen definieren
O45	Dokumentation durch Dritte prüfen lassen
O46	Funktionalitäten im Code besonders sauber dokumentieren
O47	Neu erstellte Lösungsbausteine für Anwendende dokumentieren
O48	Individualisierte Schnittstellen auf Zulässigkeit prüfen
O49	Extra IT-Sicherheit (z. B. Penn Testing) für geschäftskritische Anwendungen implementieren
O50	Anforderungsdefinition durch erfahrene Entwickler*innen und Endnutzer*innen überprüfen lassen
T1	Low-Code-Plattform in Technologiearchitektur integrieren
T2	Standardisierte Schnittstellen implementieren
T3	Designsystem mit Standard-UI-Komponenten implementieren
T4	Bereitstellungsprozess mit Versionskontrolle implementieren
T5	Forum für Support aufsetzen
T6	Zugriffsmanagement implementieren
T7	Dokumentation in den Bereitstellungsprozess integrieren
T8	Datenübertragung verschlüsseln
T9	Automatisiertes Testing implementieren
T10	Externe Schnittstellen überwachen und die Möglichkeit implementieren die Verbindung zu trennen