TU WIEN Informatics

Konzeption und prototypische Umsetzung einer No-Code-Plattform

für die Erstellung automatisierter Tests in Projekten mit Software- und Hardwarekomponenten

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Philipp Oppel, BSc.
Matrikelnummer 01427026

an der Fakultät für Informatik

der Technischen Universität Wien

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

Wien, 31. März 2025

Philipp Oppel	Thomas Grechenig

TU WIEN Informatics

Concept and Prototypical Implementation of a No-Code Platform

for Creating Automated Tests in Projects with Software and Hardware Components

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering and Internet Computing

by

Philipp Oppel, BSc.
Registration Number 01427026

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Grechenig
Assistance: Dipl. Ing. Markus Zoffi

Vienna, March 31, 2025

Philipp Oppel	Thomas Grechenig

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ich erkläre weiters, dass *ich* mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substanzielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

----------	----------
(Ort, Datum)	(Unterschrift Verfasser/in)

Kurzfassung

Diese Diplomarbeit untersucht die Konzeption und prototypische Umsetzung einer *No-Code-Plattform*, die es ermöglicht, automatisierte Systemtests für Projekte mit Software- und Hardwarekomponenten durchzuführen. Ziel ist es, Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess zu integrieren und somit die Qualitätssicherung zugänglicher und effizienter zu gestalten. Die Arbeit thematisiert die Herausforderungen bei der Automatisierung von Tests in hardwarebezogenen Projekten, insbesondere die Notwendigkeit, Fachwissen aus unterschiedlichen Bereichen zu vereinen – dazu zählen unter anderem die Komplexität der Komponenten sowie die hohe Repetitivität von Testfällen.

Im Rahmen der Arbeit wurden zwei zentrale Forschungsfragen behandelt: Zum einen die Identifikation der Anforderungen und Kriterien, die ein modellbasierter Ansatz erfüllen muss, um die Erstellung und Wartung automatisierter Tests durch Nicht-Programmierer:innen zu ermöglichen. Zum anderen die Untersuchung der Unterschiede, Limitationen und Potenziale eines solchen Ansatzes im Vergleich zu herkömmlichen Programmiermethoden. Zur Beantwortung dieser Fragen wurden systematische Literaturrecherchen, Expert:inneninterviews sowie eine prototypische Implementierung genutzt.

Der entwickelte Prototyp bietet eine Drag-and-Drop-Oberfläche, die es Fachexpert:innen erlaubt, Testfälle ohne technische Vorkenntnisse zu erstellen. Eine nahtlose Integration in CI/CD-Pipelines ermöglicht eine selbstständige Ausführung der Tests, während ein Log-Analyse-Tool Ergebnisse in verständlicher Sprache aufbereitet. Die Plattform wurde mithilfe von Expert:innenevaluierungen validiert, wobei die Rückmeldungen insbesondere den intuitiven Zugang und die Zeitersparnis durch den Wegfall manueller Programmieraufwände hervorhoben.

Die zentralen Ergebnisse der Arbeit zeigen, dass der Ansatz die Testabdeckung deutlich erhöht, indem er menschliche Interaktionen präzise automatisiert und wiederholbare Testprozesse sicherstellt. Besonders hervorgehoben wurde die verbesserte Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen, da die Plattform eine gemeinsame Grundlage für die Definition und Auswertung von Tests schafft. Durch die flexible Erweiterbarkeit des Prototyps bietet das Konzept auch Potenzial für zukünftige Anwendungen in anderen Domänen.

Trotz der positiven Ergebnisse wurden technische Limitierungen wie der erhöhte Wartungsaufwand für komplexe Testszenarien und die notwendige Weiterentwicklung der Plattform identifiziert. Die Arbeit leistet jedoch einen bedeutenden Beitrag zur Optimierung der Qualitätssicherung in software- und hardwareintensiven Projekten und verdeutlicht, wie modellbasierte Ansätze die Kluft zwischen technischer und fachlicher Expertise überbrücken können.

Keywords: *No-Code-Plattform, Testautomatisierung, Hardwareintegration, CI/CD, modellbasierte Entwicklung, FachExpert:innenintegration*

Abstract

This thesis examines the design and prototypical implementation of a no-code platform that facilitates automated system testing for projects involving software and hardware components. The goal is to integrate domain experts without programming knowledge into the development process, making quality assurance more accessible and efficient. The study addresses the challenges of test automation in hardware-related projects, particularly the need to integrate expertise from different fields, the complexity of components, and the high repetitiveness of test cases.

The thesis investigates two central research questions: first, identifying the requirements and criteria that a model-based approach must meet to enable non-programmers to create and maintain automated tests. Second, analyzing the differences, limitations, and potential of such an approach compared to traditional programming methods. To answer these questions, systematic literature reviews, expert interviews, and a prototypical implementation were conducted.

The developed prototype provides a drag-and-drop interface, allowing domain experts to create test cases without technical expertise. Seamless integration into CI/CD pipelines enables autonomous test execution, while a log analysis tool processes results in an understandable format. The platform was validated through expert evaluations, with feedback emphasizing its intuitive accessibility and time savings by eliminating manual programming efforts.

The key findings of this thesis demonstrate that the approach significantly increases test coverage by precisely automating human interactions and ensuring repeatable test processes. Improved collaboration between domain experts and developers was particularly highlighted, as the platform establishes a shared foundation for defining and evaluating tests. The prototype's flexible extensibility also offers potential for future applications in other domains.

Despite the positive results, technical limitations, such as increased maintenance effort for complex test scenarios and the need for continuous platform development, were identified. However, this thesis makes a significant contribution to optimizing quality assurance in software- and hardware-intensive projects, showcasing how model-based approaches can bridge the gap between technical and domain expertise.

Keywords: *no-code platform, test automation, hardware integration, CI/CD, model-based development, domain expert integration*

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Arbeit erforscht den Einsatz von robotergestützten Testautomatisierung (RTA) und modellbasierten Entwicklungsansätzen zur Verbesserung der Qualitätssicherung in Softwareprojekten, die sowohl Software- als auch Hardwarekomponenten beinhalten. Der Fokus liegt darauf, wie diese Technologien eingesetzt werden können, um Testverfahren nicht nur zu automatisieren, sondern auch zugänglicher und anpassungsfähiger zu gestalten, um so die Qualitätssicherung insgesamt zu verbessern. Durch die Entwicklung eines Prototyps für eine No-Code-Plattform und eines angepassten Entwicklungsprozesses zielt diese Arbeit darauf ab, Fachexpert:innen ohne Programmierkenntnisse aktiv in den Entwicklungsprozess von Software- und Hardwareprojekten einzubinden und so die bestehenden Limitationen in der Qualitätskontrolle zu überwinden.

1.1 Problemstellung

Die fortschreitende Integration der Informationstechnologie in unseren Alltag führt zu immer höheren Anforderungen an die Qualitätssicherung. Sowohl manuelle als auch automatisierte Ansätze zur Qualitätskontrolle haben ihre eigenen Vorteile und Herausforderungen. Manuelle Tests ermöglichen eine gründliche Untersuchung des Systems und seiner Funktionalitäten, sind jedoch zeitaufwendig und können durch menschliches Versagen beeinträchtigt werden [1], [2].

Automatisierte Tests können Zeit und Ressourcen sparen und eine höhere Genauigkeit erreichen, erhöhen jedoch die Komplexität und erfordern erhebliche Vorabinvestitionen [3]. Insbesondere bei Projekten, die Hardwarekomponenten beinhalten, ergeben sich aufgrund der Heterogenität der verschiedenen Komponenten zusätzliche Herausforderungen [4], [5]. Diese können den Aufbau einer Testautomatisierung erheblich erschweren und zu neuen Testfällen führen, die in einem automatisierten Aufbau nur bedingt reproduzierbar sind [6]. Eine manuelle Qualitätskontrolle kann auf Systemebene von Menschen ohne Vorkenntnisse im IT-Bereich durchgeführt werden. Es geht dabei meistens darum, definierte Schritte auszuführen und das beobachtete Verhalten zu protokollieren. Beim Automatisieren dieses Prozesses ist üblicherweise ein Maß an technischem Vorwissen notwendig, welches den Ressourcenfaktor bedeutend beeinflusst. Viele Testfälle sind zudem sehr ähnlich aufgebaut und unterscheiden sich oftmals nur in kleinen Details. Das hat für die Tester:innen ein sehr hohes Maß an Repetitivität zur Folge, welches ebenfalls Flüchtigkeitsfehler begünstigt [7].

Zur Verbesserung der Qualitätssicherung von Softwareprojekten mit Hardwarekomponenten untersucht diese Arbeit den Einsatz von robotergestützten Testautomatisierung (RTA) und No-Code-Plattformen (3.4). RTA ermöglicht die Automatisierung manueller Prozesse und kann so zur Vollautomatisierung beitragen. Allerdings führt der Einsatz von Robotern in der Software-Qualitätssicherung zu einem erhöhten Komplexitäts- und Wartungsaufwand [8].

No-Code-Plattformen könnten eine Lösung bieten, um die Komplexität zu reduzieren und die Zugänglichkeit zu erhöhen. Sie ermöglichen eine schnellere und unkompliziertere Erstellung automatisierter Tests und reduzieren den Bedarf an tiefem technischem Verständnis [9]. In diesem Beitrag wird ein Prototyp zur Entwicklung einer solchen Plattform vorgestellt.

Die Aufgabe liegt darin, Zeit und Mittel für die Qualitätssicherung bei hardwarebezogenen Projekten bestmöglich zu nutzen. Dies soll durch die Automatisierung manueller Prozesse und den Einsatz von No-Code-Plattformen erreicht werden. Ziel der Arbeit ist es, die Vorteile dieser Ansätze aufzuzeigen und zu evaluieren, ob sie eine praktikable Lösung für die Qualitätssicherungsanforderungen in komplexen Hardwareprojekten darstellen.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist ein Konzept und eine prototypische Implementierung für einen modellbasierten Ansatz für eine robotergestützte Testautomatisierung von einem System mit Abhängigkeiten zu Hardwarekomponenten. Um dieses Ziel zu erreichen, sollen folgende Forschungsfragen beantwortet werden:

Forschungsfrage F1: Welche zentralen Anforderungen und Kriterien sollten modellbasierte Ansätze erfüllen, um die effiziente Erstellung und Wartung von automatisierten Tests für unterschiedliche Testtypen durch Fachexpert:innen ohne spezifische Programmierkenntnisse zu ermöglichen?

Das Ziel dieser Fragestellung ist es, essenzielle Anforderungen und Kriterien für modellbasierte Ansätze im Bereich automatisierter Tests zu identifizieren und zu analysieren. Hierbei werden sowohl eine systematische Literaturrecherche als auch Expert:inneninterviews genutzt, um einen Überblick über den gegenwärtigen Forschungsstand und die Entwicklungen in der modellgetriebenen Softwareentwicklung zu erhalten. Diese helfen dabei, vorhandene Arbeiten, Konzepte und bewährte Praktiken zu erkennen, zu bewerten und deren Eignung für die Testautomatisierung zu bestimmen.

Im Vordergrund dieser Forschungsfrage stehen jedoch die Expert:inneninterviews, die tiefe Einblicke in die praktischen Anforderungen von Plattformen für modellbasierte Ansätze bieten. Durch die Befragung von Fachleuten und Anwender:innen wird ein umfassenderes Bild der Anforderungen erarbeitet und verdeutlicht. Die Ergebnisse der Expert:inneninterviews ergänzen und vertiefen die Erkenntnisse aus der Literaturrecherche und tragen zu einem besseren Verständnis der Schlüsselanforderungen und Kriterien bei.

In Kombination bieten die systematische Literaturrecherche und die Expert:inneninterviews eine solide Grundlage für die Ermittlung, Analyse und Bewertung der wichtigsten Anforderungen und Kriterien sowie der am besten geeignete modellbasierte Ansätze für die effiziente Entwicklung und Wartung von automatisierten Tests.

Forschungsfrage F2: Wie erleichtert ein modellbasierter Ansatz die Erstellung und Wartung von automatisierten Tests und welche Unterschiede, Limitierungen und Potenziale ergeben sich im Vergleich zu herkömmlichen Programmieransätzen?

Um Forschungsfrage F2 zu beantworten, werden eine finale Evaluierung im Rahmen einer Demonstration und ein Expert:inneninterview durchgeführt. Die finale Evaluierung, ausgeführt von unabhängigen Expert:innen, dient der Überprüfung der Funktionalität bzw. der Erfüllung der zuvor extrahierten Anforderungen der Plattform für den modellbasierten Ansatz. Diese Evaluierung ermöglicht eine objektive Bewertung des Konzepts und der technischen Umsetzung.

Das Expert:inneninterview bietet die Möglichkeit, Meinungen und Einschätzungen erfahrener Personen auf dem Gebiet einzuholen. Dabei werden gezielte Fragen zu Stärken, Schwächen, Limitationen und Optimierungsmöglichkeiten gestellt. Die Kombination aus Evaluierung und Expert:inneninterview ermöglicht ein umfassendes Verständnis von Leistungsfähigkeit und Limitationen des modellbasierten Ansatzes, um so die Forschungsfrage detailliert und fundiert zu beantworten.

Neben den Forschungsfragen, die in dieser Arbeit beantwortet werden sollen, wird die folgende Hypothese aufgestellt, die nach der Begutachtung überprüft werden soll:

Forschungshypothese: Modellbasierte Ansätze für die Erstellung und Wartung automatisierter Tests ermöglichen es Fachexpert:innen, aktiv am Entwicklungsprozess teilzunehmen. Dies erhöht die Testabdeckung und überbrückt die Lücke zwischen fachlicher und technischer Expertise.

1.3 Methodik

In diesem Abschnitt wird der methodologische Ansatz beschrieben, der für das Verfassen dieser Arbeit gewählt wurde:

Literaturrecherche

Im ersten Schritt wird eine Literaturrecherche durchgeführt, um bestehende Lösungen für dieses Problem zu finden. Im Generellen gibt es unzählbar viele verschiedene Ansätze für eine automatische Qualitätskontrolle für komplexe Software-Hardwareprojekte, jedoch sind das meist spezifische Herangehensweisen für das jeweilige Projekt, wie in [10], [11], [12] beschrieben. Die Integration von robotergestützter Qualitätssicherung mit der Entwicklung modellbasierter Ansätze zur Anwendungserstellung ist demgegenüber deutlich weniger verbreitet. Die Ergebnisse der Recherche sollen die grundsätzliche Relevanz der Thematik untermauern und das Potenzial eines generischen Ansatzes hervorheben. Außerdem werden anhand der Literaturrecherche die zugrunde liegenden theoretischen Aspekte erarbeitet und somit auch die Forschungshypothese abgeleitet.

Anforderungsanalyse

Für die Ermittlung der Anforderungen werden Expert:innen mithilfe eines Fragebogens in einer Interviewrunde befragt, die einen hybriden Ansatz verfolgt, der sowohl qualitative als auch quantitative Methoden einbezieht. Die Ergebnisse werden analysiert und dienen als Basis für die Entwicklung eines Anforderungskatalogs für modellbasierte Entwicklungsansätze, der das finale generische Lösungskonzept stützt. Zudem bietet die Interviewphase die Möglichkeit, indirekt auf Probleme hinzuweisen, die in ähnlichen Systemen bereits bekannt sind.

Design & Konzept

Mithilfe der gesammelten Anforderungen wird ein Konzept erstellt, mit dem Ziel, ein System zu entwerfen, das intuitiv von Nicht-Programmierer:innen verwendet werden kann. Das resultierende Konzept ist der generische Ansatz, anhand dessen Implementierungen für vergleichbare Systeme geschaffen werden können.

Implementierung

Nachdem in der Designphase erstellten Konzept folgt die prototypische Implementierung einer Plattform mit dem gewählten modellbasierten Ansatz. Besonders wichtig ist dabei die Ausrichtung der Plattform auf Nicht-Programmierer:innen. Die abschließende Bewertung des Prototyps erfolgt durch ein Expert:inneninterview, das entscheidend ist, um die Forschungshypothese zu bestätigen oder zu widerlegen.

Evaluierung

Um die zuvor definierten Forschungsfragen und Hypothesen zu beantworten, wurde als Ansatz ein Expert:inneninterview gewählt.

Unter Verwendung von sowohl quantitativer als auch qualitativer Fragen im Fragebogen soll eine aussagekräftige Bewertung erhoben werden.
Vor dem Interview findet eine Präsentation der wesentlichen Anwendungsfälle statt, und durch den Einsatz eines entwickelten Prototyps erhalten die Teilnehmenden einen Einblick in die Möglichkeiten, die durch modellbasierte Entwicklungsansätze geboten werden.

1.4 Aufbau der Arbeit

Diese Arbeit beschreibt den Entwicklungs- und Evaluierungsprozess eines Prototyps einer No-Code-Plattform für die robotergestützte Testautomatisierung mit dem Ziel, Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess zu integrieren. Um die zugrunde liegenden Themen zu verstehen, bietet Kapitel 2 eine umfassende Einführung in die Software-Qualitätssicherung inklusive grundlegender Konzepte, Testmethoden und Testautomatisierung.

In Kapitel 3 werden die Grundlagen von Model-Driven Development und Behavior-Driven-Development sowie die Definitionen und Limitationen von No-Code-Plattformen und Domain-Specific-Languages behandelt. Dies schafft die Basis für die Diskussion des aktuellen Stands der Technik in Kapitel 4, das Fallstudien zu Low-Code- beziehungsweise No-Code-Plattformen und robotergestützter Testautomatisierung zusammenführt und die Kombination beider Ansätze untersucht.

Die Anforderungsanalyse in Kapitel 5 basiert auf den Erkenntnissen aus dem ersten Expert:inneninterview und legt die spezifischen Anforderungen für die Entwicklung einer No-Code-Plattform fest. Hier werden auch Kriterien zur Auswahl geeigneter Plattformen beleuchtet.

Kapitel 6 widmet sich der Entwicklung und Implementierung einer No-Code-Plattform, die es Fachexpert:innen ermöglicht, automatisierte Testfälle eigenständig zu erstellen und zu verwalten. Der Fokus liegt auf einer Drag-and-Drop-Benutzeroberfläche, die durch eine nahtlose Integration in CI/CD-Systeme ergänzt wird, um eine kontinuierliche und automatisierte Qualitätssicherung zu gewährleisten. Neben der Beschreibung der Benutzerperspektive und Systemarchitektur wird ein prototypischer Ansatz vorgestellt, der das Konzept praktisch umsetzt. Besondere Herausforderungen wie Hardwareabhängigkeiten und Risikomanagement werden ebenso thematisiert wie die Erweiterbarkeit der Plattform durch Entwickler:innen. Dieses Kapitel zeigt, wie technische und organisatorische Anforderungen zusammenwirken, um eine anwenderfreundliche und skalierbare Lösung für die Testautomatisierung zu schaffen.

In Kapitel 7 werden die Ergebnisse der Expert:inneninterviews umfassend analysiert und interpretiert. Zunächst wird in einer quantitativen Auswertung die Zustimmung zu zentralen Aspekten des Konzepts statistisch beleuchtet, gefolgt von einer qualitativen Analyse, die detaillierte Einblicke in die Meinungen und Vorschläge der Befragten bietet. Besondere Aufmerksamkeit gilt dabei der Übertragbarkeit der Ergebnisse auf unterschiedliche Kontexte sowie möglichen methodischen Verzerrungen, die kritisch hinterfragt werden.

Abschließend fasst Kapitel 8 die zentralen Erkenntnisse der Arbeit zusammen, reflektiert sie kritisch im Hinblick auf die Forschungsfragen und untersucht die Limitationen der Studie. Der Ausblick skizziert zudem praxisnahe Empfehlungen und potenzielle Forschungsfelder, um die Erkenntnisse dieser Arbeit weiter zu vertiefen und in verschiedenen Branchen anzuwenden.

2 Grundlagen der Software-Qualitätssicherung

In diesem Kapitel der Arbeit werden grundlegende Begriffe und Konzepte aus den Bereichen Software-Qualitätssicherung und Testansätze vorgestellt und erläutert. Anschließend erfolgt eine detaillierte Beschreibung relevanter Anwendungsspezifika und der in dieser Arbeit verwendeten spezifischen Technologien.

2.1 Definition

In der Software-Qualitätssicherung geht es darum, Methoden und Maßnahmen zur Gewährleistung der Qualität von Softwareprodukten festzulegen und umzusetzen. Diese Disziplin ist fundamental für den Erfolg in der Softwareentwicklung und hat sich im Laufe der Jahre durch verschiedene Standards und Forschungsarbeiten weiterentwickelt [13].

Die ISO/IEC 25000-Serie[1], auch bekannt als SQuaRE (System and Software Quality Requirements and Evaluation), zielt darauf ab, einen Rahmen für die Bewertung der Softwareproduktqualität zu schaffen. Sie ist das Ergebnis der Evolution mehrerer anderer Standards, insbesondere der ISO/IEC 25010 [14], die ein Modell für die Softwareproduktevaluation definiert, und der ISO/IEC 25040:2011 [15], die den Prozess für die Softwareproduktevaluation definiert.

Ein zentraler Aspekt der Qualitätssicherung ist die Überprüfung der Produktqualität. Dabei bieten internationale Standards wichtige Richtlinien, um dies effektiv zu gestalten. Ein besonders bedeutsamer Standard in diesem Bereich ist die ISO/IEC 25010 [14], welche die Softwarequalität definiert. Diese Norm kategorisiert Softwarequalität in mehrere Dimensionen, darunter Funktionalität, Performance, Sicherheit, Komptabilität, Verlässlichkeit, Usability, Wartbarkeit und Portierbarkeit, wie in Abbildung 1 illustriert. Die Bedeutung dieses Standards liegt in seinem umfassenden Ansatz zur Definition von Softwarequalität, der sowohl für die Entwicklung als auch für die Überprüfung von Software unabdingbar ist.

[1] https://iso25000.com/index.php/en/

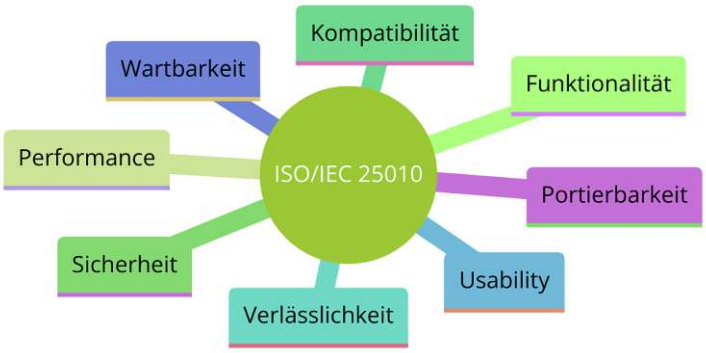

Abbildung 1 ISO/IEC 25010 Qualitätsmerkmale

Software-Qualitätssicherung umfasst verschiedene Methoden, die sich in organisatorische, statische und dynamische Ansätze unterteilen lassen [13]. Organisatorische Methoden beziehen sich auf die Einbettung von Qualitätssicherungsprozessen in den Softwareentwicklungszyklus. Diese Methoden, die unter anderem durch Boehm und Basili [16] in ihrem Werk über Softwarefehlerreduktion hervorgehoben wurden, schließen die Entwicklung von Qualitätsstandards und die Implementierung von Qualitätsmanagement-Systemen ein. Sie sind entscheidend, um eine konsistente Qualitätssicherung in der gesamten Entwicklungsphase zu gewährleisten.

Statische Methoden der Software-Qualitätssicherung, wie von McConnell in "Code Complete" [17] beschrieben, umfassen Tätigkeiten wie Code-Reviews und statische Code-Analyse, die ohne die Ausführung der Software durchgeführt werden. Diese Methoden sind von Vorteil, um Fehler im Code frühzeitig zu erkennen und zu korrigieren. Ein weiteres wichtiges Konzept in der Qualitätssicherung ist die Code-Coverage, welche misst, wie viel Prozent des Codes durch Tests abgedeckt werden. Eine hohe Code-Coverage hilft dabei, ungetestete Bereiche zu identifizieren und die Testabdeckung zu verbessern, was letztlich zur Erhöhung der Softwarequalität und zur Reduktion von Fehlern beiträgt [18]. Statische Analysetools spielen dabei eine Schlüsselrolle, da sie komplexere Fehlermuster identifizieren können, die bei manuellen Reviews möglicherweise übersehen werden.

Dynamische Methoden der Software-Qualitätssicherung hingegen beinhalten Tests, die während oder nach der Ausführung der Software durchgeführt werden. Hierzu zählen Unit-Tests, Integrationstests und Systemtests (siehe 2.5), die von Myers et al., im Werk „The Art of Software-Testing“ [19] im Bereich des Softwaretestens hervorgehoben wurden. Diese Methoden sind unerlässlich, um die korrekte Funktionsweise der Software in realen Anwendungsszenarien zu gewährleisten.

Insgesamt ist die Software-Qualitätssicherung ein multidisziplinäres Feld, das durch eine Kombination aus organisatorischen, statischen und dynamischen Methoden charakterisiert wird. Die Einhaltung von Standards wie ISO/IEC 25010 [14] und die Anwendung von bewährten Methoden aus der Literatur tragen dazu bei, die Qualität der Software zu verbessern. Es ist jedoch wichtig zu beachten, dass eine umfangreiche Qualitätssicherung nicht zwangsläufig zu guter Software führt. Dies wird auch durch die ISTQB-Prinzipien [20] unterstrichen, die besagen, dass Qualitätssicherung allein nicht ausreicht, um den Erfolg des Produkts zu garantieren [21].

2.2 Motivation

Im digitalen Zeitalter ist die Qualität von Software zu einem entscheidenden Faktor für den Erfolg von Unternehmen und Volkswirtschaften geworden. Der Bericht des Consortium for Information & Software-Quality (CISQ) aus dem Jahr 2022 [22] mit dem Titel "The Cost of Poor Software Quality in the US" (Die Kosten schlechter Softwarequalität in den USA) zeigt die finanziellen Auswirkungen minderwertiger Software deutlich auf.

Der CISQ-Bericht [22], schätzt, dass die Kosten für schlechte Softwarequalität in den USA auf 2,41 Billionen US-Dollar gestiegen sind, wobei sich die kumulierten technischen Softwareschulden (Technical Debt, TD) auf etwa 1,52 Billionen US-Dollar belaufen, was ein nicht-proportionaler Anstieg gegen über dem Jahr 2020 ist und somit nahelegt, dass das Problem immer gravierender wird. Technical Debt ist ein Sammelbegriff für Kompromisse im Entwicklungsprozess, die es schwieriger machen, die Codebasis in Zukunft zu verstehen, zu ändern und zu warten. Sie häufen sich unter anderem an, wenn Entwickler:innen Kompromisse bei Design oder Implementierung eingehen, bei denen der unmittelbare Nutzen Vorrang vor der langfristigen Qualität hat [23].

Der Bericht [22] berücksichtigt die wichtigsten wirtschaftlichen Rahmenbedingungen der USA, darunter ein für 2022 prognostiziertes BIP von 23,35 Billionen Dollar (ein Anstieg von etwa 2 % gegenüber 2020), eine Inflationsrate von 15 % im Zweijahreszeitraum, ein bescheidenes Wachstum der IT-Arbeitskräfte um 4 % in diesen zwei Jahren auf 1,51 Millionen und die Zahl der unbesetzten IT-Stellen, die Ende August bei etwa 300.000 lag.

In dem Bericht werden drei Hauptproblembereiche genannt, die zu den Kosten einer schlechten Softwarequalität beitragen:

- **Die zunehmenden Auswirkungen der Technical Debt sind zu einem der größten Hindernisse für Änderungen an bestehenden Codebasen geworden.**
- **Die Verluste durch Cyberkriminalität aufgrund bestehender Softwareschwachstellen haben erheblich zugenommen.**
- **Die Probleme in der Software-Lieferkette, insbesondere bei den zugrunde liegenden Komponenten von Drittanbietern (vor allem bei Open-Source-Software), haben drastisch zugenommen.**

Durch das Erkennen und Beheben von Software-Schwachstellen können Tests dazu beitragen, Verluste durch Cyberkriminalität zu verringern. Sie können auch dazu beitragen, Probleme mit Komponenten von Drittanbietern aufzudecken und so die Probleme in der Software-Lieferkette zu verringern [22]. Und schließlich kann regelmäßiges und gründliches Testen und Whitebox-Test beziehungsweise Code-Reviews dazu beitragen, die

Technical Debt zu verwalten und zu verringern, wodurch es einfacher wird, bestehende Code-Basen zu warten und zu verbessern [24].

Der CISQ-Bericht [22], unterstreicht den dringenden Bedarf an verbesserter Softwarequalität. Auf dem Weg in die Zukunft ist es von entscheidender Bedeutung, in umfassende Softwaretests zu investieren, um die Qualität unserer Software und damit auch die Gesundheit unserer Wirtschaft zu gewährleisten.

2.3 Definition Software-Testen

Das Testen hat in der Softwareentwicklung eine zentrale Bedeutung und wird in der Literatur vielfältig definiert [19], [25], [26]. Alle Definitionen haben gemeinsam, dass Software-Testen zum Bereich der Qualitätssicherung gehört und darauf abzielt, Softwarefehler zu identifizieren und zu beheben, die Funktionalität und Leistung der Software zu bewerten und sicherzustellen, dass sie die spezifizierten Anforderungen und Benutzererwartungen erfüllt.

Der ISTQB Foundation Syllabus [27] ist ein international anerkannter Lehrplan, der die grundlegenden Prinzipien, Techniken und Best Practices für Softwaretests vermittelt, um Fachleuten ein standardisiertes Wissen in der Qualitätssicherung von Software zu bieten. Er beschreibt in Abschnitt 1.1.1 die typischen Ziele des Testens in der Softwareentwicklung. Zu den Hauptzielen gehören das Identifizieren und frühzeitige Beheben von Fehlern, um deren Auswirkungen auf die Qualität der Software zu minimieren, und das Sicherstellen, dass die Software den Anforderungen und Spezifikationen entspricht. Tests dienen auch dazu, Vertrauen in die Software zu schaffen, indem sie deren Funktionsfähigkeit und Zuverlässigkeit bestätigen. Darüber hinaus helfen Tests, Risiken zu erkennen und zu reduzieren, die mit der Bereitstellung der Software verbunden sind. Ein weiteres Ziel des Testens ist die Bereitstellung von Informationen zur Entscheidungsfindung hinsichtlich der Freigabe der Software. Durch umfassende Tests können Entwickler:innen und Stakeholder fundierte Entscheidungen treffen und die Qualität des Endprodukts sicherstellen.

Die Leitlinien und Definitionen aus der ISO/IEC/IEEE 29119-Reihe[2] für Software-Testen legen systematische Prozesse und Techniken fest, um die Qualität von Softwareprodukten zu bewerten und zu verbessern. Software-Testen ist ein strukturierter und methodischer Prozess, der darauf abzielt, Softwareprodukte zu evaluieren. Dies umfasst die Analyse von Softwarekomponenten oder gesamten Softwarepaketen, um sowohl die Konformität mit den spezifizierten Anforderungen als auch die allgemeine Softwarequalität zu überprüfen. Software-Testen beinhaltet typischerweise das Ausführen von Software unter kontrollierten Bedingungen, um Defekte, Fehler oder andere Probleme zu identifizieren. Der Prozess wird in der Regel von Fachleuten durchgeführt und folgt standardisierten Verfahren und Richtlinien, um die Integrität und Effektivität der Tests zu gewährleisten [13].

[2] https://www.iso.org/standard/81291.html

Die Standardisierung im Bereich des Software-Testens hat an Bedeutung gewonnen, insbesondere durch die Einführung des ISO/IEC/IEEE 29119[3] Software Testing Standards. Seit ihrer Veröffentlichung im Jahr 2013 liefern diese Richtlinien einen weltweit anerkannten Rahmen für Softwaretests, der bei allen Arten von Softwareprüfungen Anwendung finden kann. Der Standard ISO/IEC/IEEE 29119[4] besteht aus mehreren Teilen, die Konzepte und Terminologie, Testprozesse, Testdokumentation und Testtechniken abdecken. Diese Standards spiegeln die aktuellen Best Practices der Industrie wider und bieten gut definierte Prozesse und Dokumentationen für die Wiederverwendung.

Software-Testen ist ein entscheidender Bestandteil des Softwareentwicklungsprozesses, der darauf abzielt, die Qualität, Sicherheit und Zuverlässigkeit von Softwareprodukten zu gewährleisten. Durch den Einsatz formeller und genehmigter Testverfahren trägt das Testen wesentlich zur Identifizierung und Behebung von Fehlern bei, was letztendlich zur Verbesserung der Gesamtqualität der Software führt [13].

2.4 Motivation von Software-Testen

Im ISTQB Foundation Syllabus [27] wird die Motivation für Softwaretests und Softwarequalitätssicherung in Kapitel 1.2 "Warum ist Testen notwendig?" behandelt. Das Testen trägt zur Kosteneffizienz bei, indem Defekte erkannt und behoben werden, und ermöglicht eine direkte Bewertung der Softwarequalität während des Entwicklungsprozesses. Tester:innen vertreten die Bedürfnisse der Benutzer:innen und helfen, vertragliche oder gesetzliche Anforderungen zu erfüllen und durch eine gezielte Ursachenanalyse von Fehlern können zukünftige Defekte vermieden oder deren Häufigkeit reduziert werden [27].

Das Testen von Software ist nicht nur ein wesentlicher Bestandteil des Entwicklungszyklus, sondern spielt auch eine entscheidende Rolle bei der Sicherstellung der Qualität und Zuverlässigkeit von Softwareprodukten. Aktuelle Studien und Forschungen im Bereich des Software-Testens bieten aufschlussreiche Einblicke in dessen Effektivität und Bedeutung [28].

Eine Studie [29] hebt hervor, wie entscheidend Software-Testen im gesamten Softwareentwicklungslebenszyklus ist. Es wird dargelegt, dass trotz des technologischen Fortschritts und der Einführung neuer Methoden in der Softwareentwicklung das Testen immer noch eine zentrale Rolle spielt. Die Autoren betonen, dass das Testen nicht nur für die Qualitätskontrolle, sondern auch für die Verifizierung, Validierung und Zuverlässigkeitsschätzung unerlässlich ist.

In der Studie "Software Testing: The State of the Practice" [30] wird ein tiefgreifender Einblick in die aktuelle Landschaft des Software-Testens gewährt, der die immense Bedeutung dieser Disziplin in der modernen Softwareentwicklung unterstreicht. Ein Schlüsselergebnis der Studie ist, dass trotz der rasanten Fortschritte in der Softwareentwicklung und dem wachsenden Druck, schnell marktreife Produkte zu liefern, das Soft-

[3] https://www.iso.org/standard/81291.html

ware-Testen oft hinterherhinkt. Besonders hervorgehoben wird die Rolle von Qualitätssicherung und Testen als kritische Faktoren, nicht nur in der Fehlererkennung und -behebung, sondern auch als entscheidende Elemente im Marketing von Softwareprodukten. Diese Erkenntnisse verdeutlichen, dass Software-Testen weit über die technische Notwendigkeit hinausgeht und ein zentraler Bestandteil der Strategie zur Sicherstellung der Produktqualität und Kundenzufriedenheit ist. Die Studie zeigt auf, wie essenziell es ist, Test-Praktiken kontinuierlich zu evaluieren und zu verbessern, um den sich ständig ändernden Anforderungen der Softwareentwicklung gerecht zu werden. Dies unterstreicht die Motivation für Software-Testen als einen unverzichtbaren Prozess, der nicht nur zur Fehlerminimierung, sondern auch zur Steigerung der Gesamteffizienz und Marktfähigkeit von Softwareprodukten beiträgt.

Diese Studien verdeutlichen, dass eine gut durchdachte und professionell umgesetzte Teststrategie entscheidend für den Erfolg eines Softwareprojekts ist. Sie belegen, dass erfahrene Test-Teams und der Einsatz moderner Test-Techniken nicht nur die Qualität der Software verbessern, sondern auch die Entwicklungszeiten signifikant verkürzen und die Effizienz steigern können.

2.5 Teststufen

Software-Tests werden in mehrere Ebenen unterteilt, wobei jede Stufe spezifische Aspekte der Softwarequalität behandelt. Eine weitverbreitete Methode zur Organisation dieser Ebenen ist das Konzept der Testautomatisierungspyramide von Mike Cohn [31]. Er beschreibt die Testautomatisierungspyramide als dreischichtiges Modell zur effektiven Teststrategie: an der Basis Unit-Tests, in der Mitte Integrationstests und an der Spitze System-Tests, siehe Abbildung 2. Unit-Tests bilden das Fundament, weil sie spezifisch und kosteneffizient sind. Die Anzahl von Systemtests, obwohl notwendig, sollte minimal gehalten werden, da sie teuer und zeitaufwendig sind. Integrationstests prüfen die Zusammenarbeit zwischen Modulen und Komponenten einer Software, wobei der Fokus je nach Ziel sowohl auf der Interaktion der Geschäftslogik als auch auf der Benutzeroberfläche liegen kann. Cohn betont, dass viele Organisationen diesen mittleren Bereich vernachlässigen, was zu teureren und weniger robusten Tests führt.

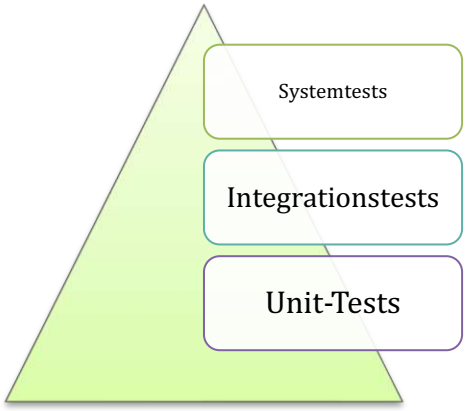

Abbildung 2 Testpyramide nach Mike Cohn [31]

Unit-Tests: Unit-Tests zielen darauf ab, die kleinsten testbaren Einheiten eines Softwareprogramms wie einzelne Funktionen oder Module isoliert von externen Abhängigkeiten zu überprüfen. In dieser Phase werden vorrangig Fehler in der Implementierungslogik oder in der Funktionalität der einzelnen Komponenten identifiziert. Dabei wird untersucht, ob jede Funktion korrekt arbeitet, die erwarteten Ergebnisse liefert und fehlerhafte Eingaben angemessen behandelt. Diese Tests sind entscheidend für die frühzeitige Erkennung von Problemen innerhalb des Codes, bevor dieser mit anderen Teilen der Software interagiert [30][31].

Integrationstests: Nachdem einzelne Module auf der Unit-Testebene verifiziert wurden, folgt die Integrationstestphase. Hier werden mehrere Module oder Komponenten kombiniert und als Gruppe getestet. Der Fokus liegt auf der Interaktion und der Datenübermittlung zwischen den Modulen. Für Integrationstests sind insbesondere die Teile ISO/IEC 29119-2:2021[5] (Testprozesse) und ISO/IEC 29119-3:2021[6] (Testdokumentation) relevant. Diese Teile bieten Richtlinien für die Planung und Durchführung von Integrationstests einschließlich der Erstellung von Testplänen und -szenarien, die sicherstellen, dass die verschiedenen Module oder Komponenten der Software korrekt zusammenarbeiten und die Daten wie vorgesehen zwischen ihnen übertragen werden [31][32].

Systemtests: Diese Tests werden auf der Ebene des vollständigen, integrierten Systems durchgeführt. Gemäß dem IEEE Std. 829-2008[7], einem Standard für Software- und Systemtestdokumentation, umfassen Systemtests die Überprüfung des gesamten Systems auf Übereinstimmung mit den spezifizierten Anforderungen. Es wird geprüft, ob das System als Ganzes funktioniert und die Benutzeranforderungen erfüllt. Dabei ist es wichtig, dass das Abbild des Systems realitätsnah ist, um beispielsweise nicht-funktionale Anforderungen (NFA) testen zu können [36].

Abnahmetests/Akzeptanztests: In dieser Phase des Softwaretests wird die Software von Endbenutzern in realen Szenarien getestet, um sicherzustellen, dass sie die definierten Anforderungen erfüllt. Dieser Prozess ist von entscheidender Bedeutung, da er die Zufriedenheit der Benutzer:innen und die Eignung der Software für den vorgesehenen Zweck bestätigt. Obwohl es keinen einheitlichen Standard für Abnahmetests gibt, bietet hier ebenfalls am ehesten IEEE Std. 829-2008[8] einen Rahmen für eine strukturierte Testdokumentation [37].

Abbildung 3 beschreibt einen wiederkehrenden Regressionszyklus im Software-Testen, der mit Unit Tests beginnt, die einzelne Funktionen und Module überprüfen. Es folgen Integrationstests, die die Zusammenarbeit zwischen verschiedenen Modulen sicherstellen. Systemtests bewerten dann das Zusammenspiel aller Teile der Software unter realistischen Bedingungen. Abschließend stellen Akzeptanztests sicher, dass die Software den Benutzeranforderungen entspricht. Nach Abschluss der Tests wird der Zyklus erneut mit Unit Tests gestartet, um sicherzustellen, dass durch neue Änderungen keine bestehenden Funktionalitäten beeinträchtigt wurden [38].

[5] https://www.iso.org/standard/79428.html
[6] https://www.iso.org/standard/79429.html
[7] https://standards.ieee.org/ieee/829/3787/
[8] https://standards.ieee.org/ieee/829/3787/

Diese Teststufen bilden das Fundament für Qualitätssicherung in der Softwareentwicklung. Sie ermöglichen eine systematische Vorgehensweise beim Testen und sind für Unternehmen und Entwickler:innen global ein entscheidender Rahmen, um die hohe Qualität und Verlässlichkeit von Software sicherzustellen [39].

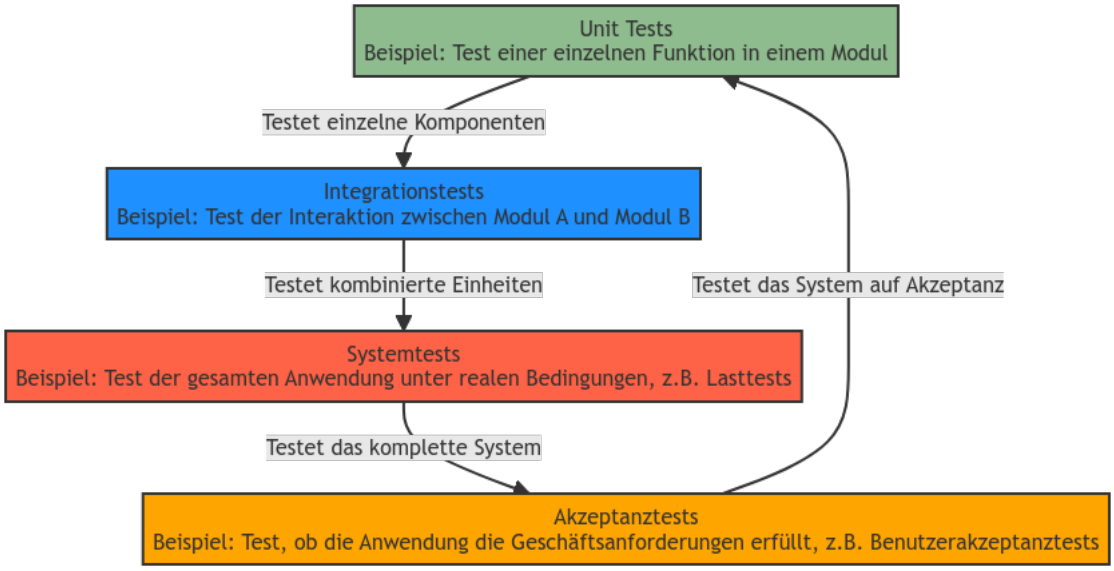

Abbildung 3 Verschiedene Testarten mit Beispielen [38]

2.6 Arten von Tests

Verschiedene Testmethoden werden angewendet, um gezielt unterschiedliche Aspekte der Softwarequalität zu evaluieren und sicherzustellen. Blackbox-Testen konzentriert sich auf die externe Funktionalität und die Benutzererfahrung der Software, ohne Einblick in den internen Code oder die Implementierungsdetails zu nehmen. Bei dieser Testmethode werden Testfälle basierend auf den Spezifikationen und Anforderungen der Software entwickelt. Tester:innen evaluieren das System, indem sie Eingaben liefern und die Ausgaben überprüfen, um sicherzustellen, dass die Software wie erwartet funktioniert. Diese Methode ist besonders nützlich, um Diskrepanzen zwischen der realen Funktionalität der Software und den ursprünglichen Anforderungen zu identifizieren. Blackbox-Testen bedient sich verschiedener Testentwurfsverfahren wie Äquivalenzklassenbildung und Grenzwertanalyse, um eine umfassende Abdeckung der Funktionalität zu gewährleisten und die Effizienz der Testbemühungen zu maximieren [40].

Whitebox-Testen taucht tiefer ein, indem es die interne Logik und Struktur des Codes untersucht. Im Gegensatz zum Blackbox-Testen, wo die inneren Arbeitsweisen der Anwendung verborgen bleiben, erfordert Whitebox-Testen den Tester:innen, Einblicke in den Code zu erhalten, um spezifische Bereiche für das Testen zu identifizieren. Diese Methode wird häufig verwendet, um die Vollständigkeit der Testabdeckung zu bewerten, indem sie sicherstellt, dass alle Pfade, Bedingungen und Schleifen im Code ausgeführt und getestet werden. Testverfahren zur Messung von Pfadüberdeckung und Zwei-

güberdeckung sind in dieser Testart von zentraler Bedeutung, um eine gründliche Analyse und Überprüfung der Software zu gewährleisten. Diese Verfahren fallen unter die kontrollflussorientierten Testverfahren [41].

Graybox-Testen stellt eine hybride Testmethode dar, die Elemente aus beiden, dem Blackbox- und dem Whitebox-Testen kombiniert. Diese Methode bietet einen Kompromiss, indem sie begrenztes Wissen über die interne Struktur der Software nutzt, um effektivere und gezieltere Testfälle zu entwerfen. Graybox -Testen ist besonders vorteilhaft, wenn es darum geht, Integrationstests und Sicherheitstests durchzuführen, da es ermöglicht, sowohl die externe Schnittstelle als auch bestimmte interne Strukturen zu evaluieren. Durch die Anwendung von Testverfahren, die auf einem begrenzten Verständnis der Softwarearchitektur basieren, können Tester:innen potenzielle Schwachstellen und Fehler effizienter identifizieren [41].

Zusammenfassend bieten Blackbox-, Whitebox- und Graybox-Testen ein umfassendes Framework für das Testen von Software aus verschiedenen Perspektiven. Während Blackbox-Testen sich auf die Benutzererfahrung und die Erfüllung der funktionalen Anforderungen konzentriert, ermöglicht Whitebox-Testen eine gründliche Überprüfung der internen Qualität des Codes. Graybox-Testen nutzt die Stärken beider Ansätze, um eine effektive und effiziente Teststrategie zu bieten. Die Wahl der Testmethode und der Testentwurfsverfahren hängt letztendlich von den spezifischen Zielen und Anforderungen des Softwareprojekts ab [13].

2.7 Testprinzipien nach ISTQB

Das International Software Testing Qualifications Board (ISTQB) [42] ist eine nicht gewinnorientierte Organisation, die sich der Förderung der Qualität und Effizienz im Bereich der Softwaretestung widmet. Durch die Standardisierung von Ausbildungsprogrammen und Zertifizierungen strebt das ISTQB danach ein hohes Niveau an Wissen und Kompetenz unter Softwaretester:innen weltweit zu etablieren. Es entwickelt umfassende Lehrpläne und führt Prüfungen durch, die darauf abzielen, Fachkräfte mit anerkannten Qualifikationen im Software-Testen zu versehen.

Die Organisation hat sieben fundamentale Prinzipien für das Software-Testen formuliert, die als Leitfaden für die effektive und effiziente Ausführung von Tests dienen. Diese zielen darauf ab, Risiken zu minimieren und Vertrauen in die Software zu schaffen. Sie bieten einen strukturierten Ansatz, um die Testprozesse zu planen, durchzuführen und zu bewerten, und helfen dabei, die Identifizierung von Defekten und die Verbesserung der Softwarequalität zu optimieren. Die Prinzipien [42] lauten wie folgt:

Testen zeigt das Vorhandensein von Fehlern: Dieses Prinzip betont, dass das Hauptziel des Testens darin besteht, Fehler zu finden. Es ist wichtig zu verstehen, dass das erfolgreiche Durchführen von Tests ohne das Auffinden von Fehlern nicht bedeutet, dass die Software fehlerfrei ist.

Vollständiges Testen ist unmöglich: Das Prinzip, dass vollständiges Testen unerreichbar ist, unterstreicht die Tatsache, dass es aufgrund der grenzenlosen Anzahl möglicher Nutzungsszenarien einer Software und ihrer Komplexität nicht machbar ist, jeden denkbaren Testfall abzudecken. Tester:innen müssen daher eine gezielte Auswahl treffen, welche Tests durchgeführt werden, indem sie sich auf die wichtigsten und wahrscheinlichsten Fehlerquellen konzentrieren. Um den Testprozess effektiver zu gestalten, setzen sie auf Testentwurfsmuster wie Grenzwerttests und Äquivalenzklassenbildung. Diese Ansätze helfen dabei, kritische Bereiche zu identifizieren, wo Fehler am ehesten auftreten, und ermöglichen es, mit einer begrenzten Anzahl von Tests ein möglichst großes Spektrum an Fehlern zu entdecken.

Frühzeitiges Testen: Je früher im Entwicklungsprozess mit Testaktivitäten begonnen wird, desto effektiver und kostengünstiger ist es. Frühzeitiges Testen hilft, Defekte und Probleme zu einem Zeitpunkt zu identifizieren, an dem sie einfacher und kostengünstiger zu beheben sind.

Häufung von Fehlern: Das Prinzip der Fehlerhäufung besagt, dass Fehler in der Softwareentwicklung nicht gleichmäßig über die gesamte Anwendung verteilt sind. Stattdessen häufen sie sich in bestimmten Bereichen oder Modulen. Derartige Muster ergeben sich oft aus der Komplexität bestimmter Funktionen, der Erfahrung des Entwicklungsteams in spezifischen Technologien oder aus der intensiven Nutzung bestimmter Komponenten durch die Endnutzer. Wenn Tester:innen und Entwickler:innen:innen diese Muster erkennen, können sie ihre Ressourcen effizienter einsetzen, indem sie mehr Aufmerksamkeit und detaillierte Tests den Teilen der Software widmen, die am anfälligsten für Fehler sind. Eine solche strategische Fokussierung ermöglicht eine gezieltere Fehlerbehebung und verbessert die allgemeine Qualität und Stabilität der Software, indem sie sicherstellt, dass die kritischsten und fehleranfälligsten Bereiche gründlich getestet und optimiert werden.

Das Pestizid-Paradoxon: Wenn die gleichen Tests wiederholt durchgeführt werden, werden schließlich keine neuen Fehler mehr gefunden. Es ist notwendig, die Testfälle regelmäßig zu überarbeiten und zu variieren, um neue Fehler aufzudecken.

Testen ist kontextabhängig: Je nach Art der Anwendung und des Projekts variieren die Anforderungen und Ansätze für das Testen. Beispielsweise unterscheiden sich die Teststrategien für eine medizinische Software deutlich, von denen für ein Videospiel.

Trugschluss des fehlenden Fehlers: Dieses Prinzip macht deutlich, dass das QS-Maßnahmen allein nicht ausreichen, um den Erfolg oder die Qualität einer Software zu garantieren. Obwohl Tests dabei helfen, Fehler zu finden, können sie nicht alle möglichen Probleme aufdecken. Faktoren wie Benutzerfreundlichkeit, Leistung oder Sicherheit können trotz erfolgreicher Tests unzureichend sein. Deshalb ist es wichtig, die Software in ihrer Gesamtheit zu bewerten, da auch ein fehlerfreies Testergebnis nicht sicherstellt, dass das Produkt die Erwartungen oder Anforderungen vollständig erfüllt.

2.8 Testprozess nach ISTQB

Der Testprozess im Kontext des ISTQB [27], [42] dient dazu, die Qualität von Softwareprodukten zu überprüfen und zu verbessern. Er umfasst systematische Aktivitäten,

die auf die Identifizierung und das Beheben von Defekten in der Software abzielen, um sicherzustellen, dass das Produkt die Anforderungen und Erwartungen der Stakeholder erfüllt.

Allgemein zielt der Testprozess darauf ab, Vertrauen in die Qualität des Produkts zu schaffen, Informationen über den Qualitätsstatus des Produkts bereitzustellen, Risiken im Zusammenhang mit der Freigabe und dem Betrieb des Produkts zu minimieren und sicherzustellen, dass das Produkt den definierten Anforderungen entspricht [43].

Er besteht aus verschiedenen Phasen, die von der Testplanung und -kontrolle über die Analyse und das Design von Tests, die Implementierung und Ausführung von Tests bis hin zum Abschluss der Testaktivitäten reichen. Jede dieser Phasen hat spezifische Ziele und Aufgaben, die darauf abzielen, Defekte zu identifizieren, die Testabdeckung zu maximieren und die Softwarequalität zu gewährleisten. Der Testprozess ist ein integraler Bestandteil der Softwareentwicklung und trägt dazu bei, dass das Endprodukt den Bedürfnissen der Nutzer entspricht und in einer sich schnell verändernden Umgebung zuverlässig funktioniert. Darüber hinaus kann der Testprozess parallel zum Entwicklungsprozess ablaufen, wodurch potenzielle Fehler frühzeitig erkannt und behoben werden können. Dies fördert eine kontinuierliche Qualitätssicherung und trägt zur Effizienz des gesamten Entwicklungszyklus bei [44].

Testplanung und -steuerung: In dieser Phase wird definiert, was getestet wird, wie die Tests durchgeführt werden, und die Ziele und der Umfang der Tests werden festgelegt. Es werden Ressourcen zugewiesen, und der Zeitplan wird erstellt. Die Testplanung ist ein kontinuierlicher Prozess, der während des gesamten Testprozesses fortgeführt wird, um sicherzustellen, dass die Testaktivitäten im Einklang mit den Zielen stehen und effizient gesteuert werden.

Testanalyse und -design: Während dieser Phase werden die Testbedingungen analysiert und die Testfälle entworfen. Dies beinhaltet das Verstehen der Testbasis (wie Anforderungsspezifikationen, Risikobewertung, Architekturdiagramme usw.) und das Ableiten von Testfällen aus diesen Informationen.

Testimplementierung und -ausführung: In dieser Phase werden die Testfälle in eine logische Reihenfolge gebracht und Testprozeduren oder Testskripte vorbereitet. Die Testumgebung wird eingerichtet, und alle notwendigen Testdaten werden generiert oder angepasst. Während der Testdurchführungsphase werden die Testfälle ausgeführt, und die Ergebnisse werden mit den erwarteten Ergebnissen verglichen. Abweichungen werden als Defekte aufgezeichnet. Diese Phase beinhaltet auch das Wiederholen von Testfällen nach Defektkorrekturen.

Evaluierung der Exit-Kriterien und Berichterstattung: In dieser Phase wird überprüft, ob die festgelegten Ziele erreicht wurden und ob die Software das Ende der Testphase erreicht hat. Die Ergebnisse der Testaktivitäten werden zusammengefasst und berichtet, was Entscheidungen über die Freigabe der Software unterstützt.

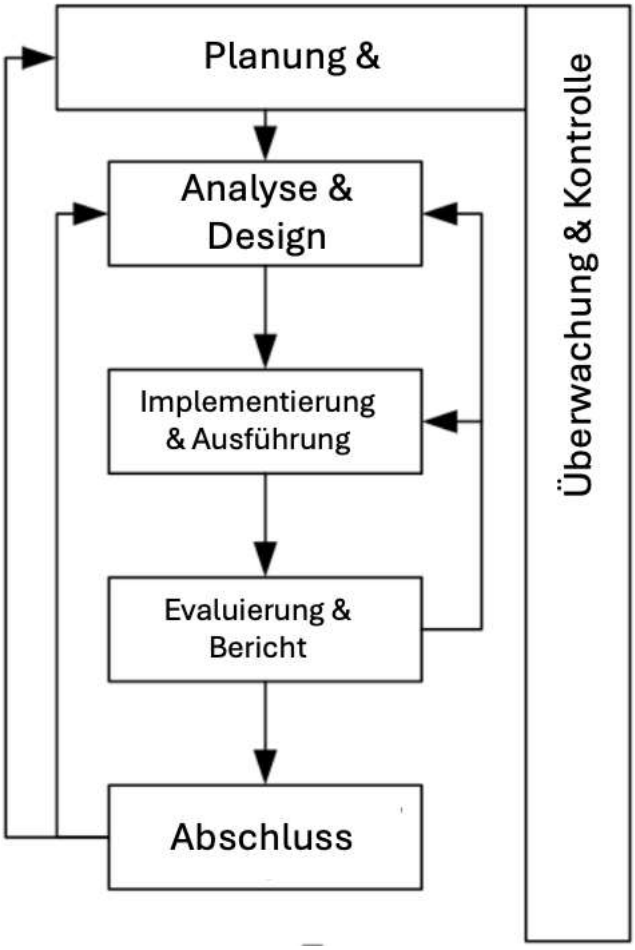

Abbildung 4 Visualisierung des Testprozesses gemäß ISTQB [39]

Testabschluss: Die Testabschlussaktivitäten umfassen das Sammeln von Daten aus abgeschlossenen Testaktivitäten zur Verbesserung zukünftiger Testprozesse. Es geht um die Archivierung von der Auswertung des Testprozesses und das Erstellen eines Testabschlussberichts.

In Abbildung 4 wird der Prozess visualisiert, der den Ablauf und die Zusammenhänge zwischen den oben beschriebenen Schritten darstellt. Jede dieser Phasen ist entscheidend für den Gesamterfolg des Testprozesses. Sie ermöglichen eine strukturierte Herangehensweise an das Testen, die sowohl die Identifikation von Defekten als auch die kontinuierliche Verbesserung des Testprozesses selbst unterstützt. Der iterative Ansatz des ISTQB-Testprozesses erlaubt es, Anpassungen vorzunehmen und die Testaktivitäten fortlaufend an die sich ändernden Anforderungen des Projekts anzupassen [42].

2.9 Testautomatisierung

Testautomatisierung ist ein entscheidendes Konzept in der Softwareentwicklung, insbesondere zur Sicherstellung der Qualität und Effizienz von Softwareprodukten und zählt zu den dynamischen Methoden der Software-Qualitätssicherung (siehe 2.1). Genauer gesagt, ergänzt die Testautomatisierung den manuellen Testprozess, indem dadurch Testaufgaben automatisiert ablaufen. Dabei wird spezialisierte Software genutzt, um

Tests systematisch auszuführen und die tatsächlichen Ergebnisse mit den vordefinierten Erwartungen abzugleichen. Dies ermöglicht eine effiziente und präzise Überprüfung der Softwarequalität, ohne dabei die Notwendigkeit einer menschlichen Beurteilung in komplexen oder nicht standardisierten Testfällen zu eliminieren [45].

Laut ISTQB [42] bestehen die Hauptziele der Testautomatisierung darin, den manuellen Arbeitsaufwand im Testprozess zu reduzieren, die Testabdeckung und -effizienz zu erhöhen und dem Entwicklungsteam schnelles Feedback zu geben [13].

Ein weiteres wichtiges Konzept in diesem Zusammenhang ist die kontinuierliche Integration oder Continuous Integration (CI). CI bezeichnet einen Entwicklungsprozess, bei dem der individuelle Code der Entwickler:innen regelmäßig in eine gemeinsame Codebasis integriert wird. Dieser Ansatz zielt darauf ab, Integrationsprobleme frühzeitig zu erkennen und zu beheben, indem jede Integration unmittelbar durch eine Reihe automatisierter Tests überprüft wird [46].

Continuous Deployment (CD) erweitert das Konzept der Continuous Integration. Nach erfolgreicher Integration mit Testdurchläufen in der CI-Phase wird der Softwarecode automatisch in die Produktionsumgebung überführt. Diese automatisierte Bereitstellung stellt sicher, dass die Software stets in einem auslieferungsfähigen Zustand bleibt und ermöglicht eine schnelle Freigabe von Funktionsupdates und Fehlerbehebungen. Das CD-Modell ist besonders vorteilhaft in dynamischen Entwicklungsumgebungen, in denen schnelle Iterationen und schnelle Reaktionen auf Marktänderungen erforderlich sind [47].

Durch die Automatisierung routinemäßiger Testaufgaben ermöglicht die Testautomatisierung eine systematische und präzise Prüfung der Softwarequalität, wobei sie gleichzeitig die Grenzen manueller Tests durch menschliche Beurteilung in komplexen oder einzigartigen Testszenarien anerkennt [45].

Die von ISTQB bereitgestellten Testprinzipien [27] bilden eine wichtige Grundlage für das Verständnis und die Implementierung von Testautomatisierungsstrategien. Diese Strategien zielen darauf ab, den manuellen Aufwand zu minimieren, die Testabdeckung und -effizienz zu verbessern und schnelles Feedback für Entwicklungsteams zu generieren [45].

CI fördert die regelmäßige und problemlose Integration individueller Codebeiträge in eine gemeinsame Basis, während CD den Prozess durch die automatische Überführung des getesteten Codes in die Produktionsumgebung weiterführt. Diese Prozesse sorgen für eine ständige Bereitschaft der Software zur Auslieferung und unterstützen agile Entwicklungspraktiken, die schnelle Iterationen und Anpassungen an Marktveränderungen erfordern [46][47].

Obwohl die Testautomatisierung für moderne Software-Entwicklungsprozesse unerlässlich ist, birgt sie spezifische Herausforderungen. Erstens kann die Einrichtung automatisierter Tests sehr zeitaufwendig und teuer sein. Um Tests effektiv in CI/CD-Pipelines zu integrieren, ist eine sorgfältige Planung und Einrichtung der Testumgebungen insbesondere für Systemtests erforderlich. Die fortlaufende Wartung, wie in Kapitel 2.7 beschrieben, sorgt für zusätzlichen Aufwand: Bei jeder Änderung am Produkt müssen

Tests angepasst oder neu entwickelt werden, um Softwareabdeckung zu gewährleisten und relevante Tests zu erhalten. Eine weitere Hürde sind falsch positive oder falsch negative Ergebnisse, die durch instabile Testumgebungen oder schlecht konzipierte Testfälle entstehen können. Sie können das Vertrauen in die Testautomatisierung beeinträchtigen und zusätzliche manuelle Prüfungen erforderlich machen, wodurch der zeitliche und Kostenvorteil der Automatisierung reduziert wird. Dazu kommt die Schwierigkeit, qualifiziertes Personal zu finden. Die Entwicklung und Wartung automatisierter Testszenarien erfordern Fachkräfte, die über Know-how in Softwareentwicklung und Testdesign verfügen. Fehlende Fachkräfte können die Wirksamkeit der Testautomatisierung beeinträchtigen. Trotz dieser Herausforderungen bleibt die Testautomatisierung entscheidend für die Softwareentwicklung. Bei richtiger Umsetzung steigert sie die Effizienz und trägt dazu bei, dass Software-Entwicklungsprojekte hohe Qualitätsstandards einhalten. Ein weiterer positiver Aspekt ist die Regressionssicherheit, die durch regelmäßige Regressionsprüfungen gewährleistet wird. Diese Tests stellen sicher, dass neue Änderungen oder Aktualisierungen keine bestehenden Funktionen beeinträchtigen, was zur Stabilität und Zuverlässigkeit der Software beiträgt [48].

2.10 Versionskontrolle

Continuous Integration und Continuous Deployment (CI/CD) sind zentrale Methoden in der modernen Softwareentwicklung, die auf Versionskontrollsystemen basieren. Sie ermöglichen automatisierte Tests und Bereitstellungen für jede Codeänderung, wodurch Neuerungen kontinuierlich integriert, getestet und in die Produktionsumgebung überführt werden können [49].

Die Versionskontrolle ist ein System, das alle Änderungen an den Dateien eines Projekts verfolgt und dokumentiert. Ein Branch erlaubt es, eine unabhängige Entwicklungslinie vom Hauptzweig des Projekts zu erstellen. Dies ermöglicht die parallele Entwicklung von neuen Funktionen oder Fehlerbehebungen, ohne den stabilen Hauptcode zu beeinträchtigen. Beim Auschecken (Checkout) wird ein bestimmter Branch oder eine Version des Projekts lokal kopiert, um daran zu arbeiten. Änderungen werden durch Committen lokal gespeichert und mit einer Beschreibung versehen, bevor sie hochgeladen werden. Diese Praktiken fördern eine strukturierte und sichere Zusammenarbeit an Projekten [49].

Durch die Integration von Versionskontrolle und CI/CD entsteht ein effizienter und zuverlässiger Entwicklungsprozess. Automatisierte Tests und Bereitstellungen stellen sicher, dass Änderungen kontinuierlich und sicher in die Produktion überführt werden. Dies erhöht die Softwarequalität, beschleunigt den Entwicklungszyklus und verbessert die Zusammenarbeit innerhalb des Teams [49].

2.11 Robotergestützte Testautomatisierung

In bestimmten Testumgebungen sind Automatisierungen ohne den Einsatz von Robotern nicht durchführbar. Dies betrifft insbesondere Szenarien, bei denen die Software stark von der Interaktion mit Hardware abhängt. Ohne Roboter ist es kaum möglich,

präzise und wiederholbare Tests durchzuführen, da menschliche Eingriffe oft ungenau und fehleranfällig sind. Roboter ermöglichen es, komplexe Bewegungsabläufe oder Hardware-Interaktionen zu absolvieren und so Szenarien zu testen, die durch reine Softwareautomatisierung nicht erfasst werden können [50].

In modernen Testumgebungen stellt die Testautomatisierung mithilfe von Roboterarmen, auch bekannt als Robotic Test Automation (RTA), eine zunehmend beliebte Methode dar, um Effizienz, Präzision und Wiederholbarkeit von Testprozessen zu verbessern [53]. Diese Technologie nutzt mechanische Arme, die programmierbar sind, um eine Vielzahl von Aufgaben automatisch auszuführen, die traditionell von Menschen durchgeführt wurden. Solche Automatisierungssysteme sind besonders in Branchen wie der Automobilindustrie [54], Softwareentwicklungsprojekte mit Hardwarekomponenten [55], und vielen weiteren Industriegebieten [56], wo sie zur Qualitätssicherung und zur Beschleunigung von Entwicklungszyklen beitragen.

Die robotergestützte Testautomatisierung stellt einen bedeutenden Fortschritt in der Softwarequalitätssicherung dar, indem sie die Effizienz und Genauigkeit von Testprozessen erheblich verbessert. Durch den Einsatz von Roboterarmen können wiederholbare und präzise Tests durchgeführt werden, die menschliche Fehler minimieren und die Zuverlässigkeit der Ergebnisse erhöhen. Roboterarme ermöglichen es, komplexe Interaktionen mit Hardwarekomponenten zu simulieren und so realistische Testumgebungen zu schaffen, die für die Validierung von Software unerlässlich sind. Durch die Integration von Roboterarmen in die Testautomatisierung können Unternehmen die Qualität ihrer Softwareprodukte erheblich steigern und gleichzeitig die Zuverlässigkeit und Genauigkeit der Tests erhöhen, indem menschliche Fehler minimiert und realistische Hardware-Interaktionen präzise simuliert werden [44][45].

Ein wesentlicher Vorteil vom Einsatz von Roboterarmen ist die Fähigkeit, Systemtests auf realer Basis ohne Virtualisierung von Hardwarekomponenten durchzuführen. Dadurch können reale Bedingungen simuliert werden, was zu aussagekräftigeren Testergebnissen führt [52].

Trotz der Vorteile sind mit der Automatisierung durch Roboterarme auch Herausforderungen verbunden. Zusätzlich zu allgemeinen Faktoren wie begrenzten Ressourcen und der Notwendigkeit der strategischen Ausrichtung digitaler Initiativen gibt es spezifische Probleme bei der Ausführung. Roboterarme können möglicherweise nicht alle Aktionen präzise durchführen, was zu Fehlern und unzuverlässigen Testergebnissen führen kann. Sogenannte „False Positives“ können auftreten, wenn ein Test fälschlicherweise ein Problem meldet, obwohl keines vorhanden ist, z. B. durch ungenaue Platzierung von Gegenständen durch den Roboter. Außerdem kann die Kontrolle und Synchronisation der Roboterbewegungen komplex und fehleranfällig sein, was zu Kettenreaktionen und Ausfällen in vernetzten Systemen führen kann [53]. Auch die genaue Simulation menschlicher Interaktionen und die Notwendigkeit umfangreicher Programmierung und Kalibrierung stellen Herausforderungen dar. Roboterarme benötigen eine hohe Genauigkeit und Präzision bei der Identifizierung der Berührungspunkte. Das Setup erfordert detaillierte Programmierung und umfangreiche Kalibrierung, um sicherzustellen, dass die Tests realistische Bedingungen abbilden. Zudem müssen umfangreiche Statistiken zu Berührungen, Genauigkeit, Zeitstempeln und Fehlerquoten erhoben und analysiert

werden, was den Entwicklungsprozess erheblich komplizieren kann [54]. Eine strategische Planung ist essenziell, um diese Herausforderungen zu bewältigen und digitale Technologien erfolgreich zu integrieren [8].

Darüber hinaus müssen neben den erhöhten Personalkosten auch die höheren Investitionskosten für die erforderliche Infrastruktur und die Roboter selbst berücksichtigt werden. Eine initiale Analyse zur Bestimmung der Amortisationszeit der Kosten ist entscheidend. Eine Studie [55] deutet auf einen negativen wirtschaftlichen Effekt aufgrund der Kosten für Verbrauchsmaterialien hin, während eine andere Studie [56] aufzeigt, dass in einem Industrieprojekt die Kosten sich innerhalb von weniger als 2,5 Jahren amortisieren können.

Trotz dieser Herausforderungen überwiegen die Vorteile der Testautomatisierung mit Roboterarmen deutlich. Die Technologie bietet nicht nur eine verbesserte Testeffizienz und -genauigkeit, sondern auch eine größere Flexibilität. Die Fähigkeit, Tests schnell und präzise zu wiederholen, stellt einen signifikanten Fortschritt in der Qualitätssicherung dar. Durch die Integration von Roboterarmen in die Testautomatisierung können Unternehmen die Abdeckung realistischer Szenarien erweitern und die Interaktion mit Hardware präzise steuern. Roboter dienen als wertvolle Ergänzung zur Virtualisierung, indem sie sicherstellen, dass hardwareabhängige Tests effizient und unter realen Bedingungen durchgeführt werden. So wird die Qualität der Softwareprodukte erhöht, insbesondere durch die zuverlässige Validierung von Funktionen, die in reinen Softwareumgebungen nicht erfasst werden können. Die zunehmende Integration von Roboterarmen in Testautomatisierungsprozesse ist daher ein klarer Indikator für den fortschreitenden Trend zur Automatisierung in der modernen Qualitätssicherung [57].

2.12 Robotersteuerung

Die Robotik kombiniert Technologien aus der Mechanik, Elektronik und Informatik, um Maschinen zu entwickeln, die Aufgaben autonom oder halb-autonom erledigen können. Dieses Kapitel gibt einen generellen Überblick über die Grundlagen, welche für den Rahmen dieser Arbeit relevant sind.

Ein typischer Roboter, wie zum Beispiel ein Greifarm, setzt sich aus mehreren wesentlichen Komponenten zusammen. Zuerst ist die mechanische Struktur entscheidend, die aus Gelenken und Gliedern besteht. Diese Struktur ermöglicht es dem Roboter, Bewegungen in mehreren Freiheitsgraden auszuführen, was für Aufgaben wie Greifen und Manipulation notwendig ist [58].

Aktoren sind die Komponenten, die elektrische Signale des Steuerungssystems in physische Bewegungen umsetzen. Diese können entweder in Form von Elektromotoren, hydraulischen oder pneumatischen Antrieben vorliegen und ermöglichen die Bewegung und Kraftausübung, die für die Manipulation der Umwelt nötig sind. Das Steuerungssystem eines Roboters ist der "Gehirn"-Teil der Maschine. Hier kommen Mikrocontroller und

Mikroprozessoren zum Einsatz, die Bewegungen planen und die Aktoren steuern. Beliebte Steuerungseinheiten umfassen Mikrocontroller wie den Arduino[9] oder leistungsfähigere Prozessoren wie den NVIDIA Jetson[10], die besonders in komplexen Robotersystemen verwendet werden [59].

Eine wesentliche Rolle bei der Steuerung von Robotern spielt das Robot-Operating-System (ROS) [60]. Dieses Open-Source-Framework ermöglicht die modulare Programmierung von Robotersystemen, indem es eine Sammlung von Bibliotheken und Tools bereitstellt, die es Entwickler:innen erlauben, verschiedene Hardware- und Softwarekomponenten zu integrieren [61]. Beispielsweise bietet ROS APIs für Programmiersprachen wie Python[11], wodurch Roboter flexibel und effizient angesteuert werden können. Diese Verbindung vereinfacht die Entwicklung von Robotersystemen, indem sie eine direkte und effiziente Schnittstelle für die Steuerung und Programmierung von Robotern bietet. Dadurch kann schnell auf verschiedene Funktionen des Roboters zugreifen, ihn steuern und an individuelle Aufgaben anpassen [62].

Durch die Verwendung von ROS in Verbindung mit Programmiersprachen wie Python wird eine flexible und effiziente Entwicklung ermöglicht, die eine einfache Anpassung an unterschiedliche Aufgaben erlaubt [61], [62]. Diese Technologien und Konzepte sind nicht nur für die in dieser Arbeit behandelten Beispiele relevant, sondern bieten auch ein übertragbares Fundament für zahlreiche weitere Anwendungsbereiche in der Robotik und Automatisierung.

[9] https://www.arduino.cc
[10] https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/
[11] https://www.python.org

3 Grundlagen von Model-Driven Development

Dieses Kapitel bietet eine einführende Untersuchung des Konzepts des Model-Driven-Development (MDD), das darauf abzielt, die Softwareentwicklung zu vereinfachen und zu beschleunigen. Die Diskussion umfasst eine Untersuchung seiner grundlegenden Prinzipien, Vorteile und seiner zentralen Rolle in der aktuellen und zukünftigen IT-Landschaft. Darüber hinaus wird untersucht, wie MDD die Softwareentwicklung verbessert hat, indem es eine höhere Abstraktionsebene bietet, was es auch weniger technisch versierten Personen ermöglicht, sich aktiv am Entwicklungsprozess zu beteiligen. Letztlich soll dieses Kapitel veranschaulichen, wie MDD das Paradigma der Softwareentwicklung umgestaltet und warum es zu einem unverzichtbaren Werkzeug in der IT-Branche geworden ist.

3.1 Definition

Model-Driven Development (MDD) ist ein vielseitiger Entwicklungsansatz, der eine breite Palette von modellbasierten Methoden und Technologien umfasst. MDD bietet eine zusätzliche Abstraktionsschicht, die es ermöglicht, sowohl gesamte Systeme als auch spezifische Komponenten effizient zu gestalten und zu warten. Durch die flexible Anwendung von Modellen auf unterschiedliche Systemteile wird die Wartbarkeit erhöht, während die Entwicklungsansätze auf die spezifischen Herausforderungen jeder Komponente oder des Gesamtsystems abgestimmt werden. Dies erlaubt eine gezielte Optimierung sowohl für vollständige Systeme als auch für einzelne Systembestandteile [63]. Dabei reicht MDD von der Verwendung von Design-Patterns über die Modellierung von Geschäftsprozessen bis hin zur vollständigen Automatisierung von Code-Generierung [64].

Programmier- und Entwurfsmuster (Design-Pattern) spielen in MDD eine zentrale Rolle, beispielsweise das Page-Object-Pattern im Bereich des UI-Testings. Hierbei wird die Funktionalität einer Webseite als Modell (Page-Object) dargestellt, das die Struktur und Interaktionen der Benutzeroberfläche abstrahiert. Dies ermöglicht es, Änderungen an der UI unabhängig vom zugrunde liegenden Code vorzunehmen, was die Wartbarkeit und Erweiterbarkeit verbessert [65].

Ein weiteres Beispiel für die Anwendung von MDD ist Business Process Model and Notation (BPMN), ein Standard zur Modellierung von Geschäftsprozessen. BPMN-Diagramme ermöglichen es, komplexe Abläufe in einer für Fachabteilungen standardisierten und verständlichen Darstellung zu modellieren, was die Zusammenarbeit zwischen Entwickler:innen und Fachexpert:innen erheblich erleichtert [64].

Behavior-Driven Development (BDD) ist ein weiteres Schlüsselelement im MDD-Ansatz (nähere Details in Kapitel 3.3). BDD fördert die Schaffung einer gemeinsamen Sprache zwischen Entwickler:innen, Testern und Fachabteilungen. Anstatt auf technische Details einzugehen, werden Anforderungen in einer klaren, verständlichen Sprache formuliert, die anschließend in automatisierte Tests und Implementierungen umgewandelt werden können. Diese Methode erleichtert die Kommunikation und sorgt dafür, dass alle Beteiligten ein einheitliches Verständnis der Anforderungen haben [66].

Neben BDD spielen Domain-Specific Languages (DSLs) eine wichtige Rolle. DSLs sind spezielle Sprachen, die für einen bestimmten Anwendungsbereich entwickelt wurden und es Entwickler:innen ermöglichen, auf einer höheren Abstraktionsebene zu arbeiten, ohne sich um die Details der technischen Implementierung kümmern zu müssen. Sie erleichtern es, komplexe Domänenlogiken zu modellieren und anschließend automatisiert in Code zu übersetzen [67].

Low-Code- und No-Code-Plattformen sind ebenfalls wichtige Ansätze von MDD. Sie bieten visuelle Entwicklungsumgebungen, in denen Anwendungen durch einfache Drag-and-Drop-Techniken oder vereinfachtem Code erstellt werden können. Diese Plattformen senken die technische Hürde und vereinfachen das Entwickeln und Warten von Software. Dadurch kann eine schnellere Entwicklung und Wartung von Anwendungen möglich sein, indem die manuelle Codierung reduziert wird [68].

Zusammenfassend lässt sich sagen, dass MDD eine Vielzahl von Methoden und Technologien umfasst, die es ermöglichen, komplexe Softwareentwicklungsprojekte durch den Einsatz von Modellen auf verschiedenen Abstraktionsebenen zu vereinfachen. Ob durch die Verwendung von Design-Patterns wie dem Page-Object-Model, der Modellierung von Geschäftsprozessen mit BPMN, der Nutzung von BDD oder DSLs – MDD bietet zahlreiche Möglichkeiten, um sowohl die Verständlichkeit als auch die Wartbarkeit von Software zu verbessern und gleichzeitig die technische Implementierung zu automatisieren und zu optimieren.

3.2 Motivation

Model-Driven Development (MDD) ist eine moderne Herangehensweise an die Softwareentwicklung, die den Fokus auf die Erstellung und Nutzung von Modellen legt, um die Entwicklung von Systemen und Anwendungen zu erleichtern [69]. Diese Methode bietet besonders für Fachexpert:innen ohne tiefgehende Programmierkenntnisse erhebliche Vorteile.

Die Studie [70] zeigt, wie MDD in der Industrie genutzt werden kann, um sogenannte "Digitale Zwillinge" zu entwickeln. Ein Digitaler Zwilling ist eine virtuelle Darstellung eines physischen Produkts oder Systems, die in Echtzeit aktualisiert wird und somit eine genaue Überwachung und Steuerung ermöglicht. Der modellgetriebene Ansatz hilft IndustrieExpert:innen, den Entwicklungsprozess dieser digitalen Zwillinge zu steuern, ohne dass tiefgehende Programmierkenntnisse erforderlich sind. Dadurch können Fachexpert:innen ihre spezifischen Kenntnisse direkt einbringen und maßgeschneiderte Lösungen entwickeln. Die Studie beschäftigt sich mit einem modellgetriebenen Ansatz für

die wissensbasierte Entwicklung von industriellen digitalen Zwillingen. Dabei werden abstrakte Modellierungssprachen wie SysML[12] oder UML[13] verwendet, um virtuelle Repräsentationen physischer Systeme zu erstellen. Die Modelle werden mit speziellem Fachwissen erweitert, das als Regeln und Vorgaben festgelegt wird. Zum Beispiel könnte eine Regel sein, dass eine Maschine gestoppt wird, wenn sie überhitzt. Dieses Wissen wird durch Bausteine eingebunden, die in Echtzeit auf die Daten des physischen Systems zugreifen und basierend auf den Regeln, wie etwa dem Temperaturwert der Maschine, automatisch Entscheidungen treffen. Das Ziel ist es, die Entwicklung und Wartung von digitalen Zwillingen zu vereinfachen und zu automatisieren, indem sowohl Modellierung als auch Fachwissen kombiniert werden. Die vorgestellte Methodik ist anpassbar und übertragbar auf verschiedene industrielle Systeme [70].

In [71] wird untersucht, wie MDD die Entwicklung von Prototypen für das Internet der Dinge (engl. Internet of Things, IoT) erleichtern kann, besonders im Bereich der intelligenten Landwirtschaft. Sie zeigen, dass Landwirte und andere Fachexpert:innen komplexe IoT-Anwendungen erstellen können, ohne Programmierkenntnisse zu benötigen. Dies geschieht durch die Verwendung von vordefinierten Bausteinen, die flexibel kombiniert werden können, um spezifische Lösungen zu entwickeln, die genau auf die Bedürfnisse der Nutzer zugeschnitten sind. Ein Landwirt, der ein System zur Überwachung der Bodenfeuchtigkeit und der Wetterbedingungen einrichten möchte, kann mit MDD diese Sensoren und Überwachungsgeräte mithilfe vorgefertigter Module kombinieren, um ein maßgeschneidertes Überwachungssystem zu erstellen, das ihm hilft, die Bewässerung seiner Felder effizienter zu gestalten, ohne dass er dafür programmieren muss [71].

Die Nutzung von speziellen Modellierungssprachen und unterstützenden Werkzeugen, die den Modellierungsprozess erheblich erleichtern, wurden in [72] untersucht. Diese Werkzeuge wie Debugger und Testtools machen den Prozess der Softwareentwicklung zugänglicher für Fachexpert:innen mit minimalen Programmierkenntnissen. Die Studie zeigt, dass diese Herangehensweise komplexe Systeme effizient und fehlerfrei entwickeln kann, was die Beteiligung von Expert:innen aus verschiedenen Fachgebieten an der Softwareentwicklung fördert [72].

Model-Driven Development bietet eine effektive Möglichkeit, die Kluft zwischen technischer und fachlicher Expertise zu überbrücken [70] [71] [72]. Durch die Verwendung von Modellen und speziellen Modellierungssprachen können Fachexpert:innen direkt an der Entwicklung und Anpassung von Softwarelösungen teilnehmen, ohne dass tiefgehende Programmierkenntnisse erforderlich sind. Dies fördert nicht nur die Effizienz und Qualität der entwickelten Systeme, sondern ermöglicht auch eine engere Zusammenarbeit zwischen verschiedenen Expert:innen und Disziplinen.

[12] https://sysml.org
[13] https://www.uml.org

3.3 Behavior-Driven-Development / Behavior-Driven-Testing

Behavior-Driven-Development (BDD) ist eine Softwareentwicklungsmethode, die darauf abzielt, das Verständnis und die Kommunikation zwischen Entwickler:innen, Qualitätssicherungsteams und nicht-technischen Stakeholdern zu verbessern. BDD konzentriert sich auf die Erstellung von Software durch beschreibende Sprache, die das gewünschte Verhalten der Software aus der Perspektive des Endnutzers definiert. Diese Methode hat ihre Wurzeln im Test-Driven Development (TDD), einer Praxis, bei der Entwickler:innen zuerst Tests schreiben, bevor die Implementierung eines Features begonnen wird. Der entscheidende Unterschied bei BDD liegt darin, dass es einen Schritt weiter geht, indem es die Bedeutung der Sprache und des Dialogs zwischen den Stakeholdern hervorhebt, um sicherzustellen, dass die entwickelte Software genau den Bedürfnissen und Erwartungen des Kunden entspricht [73].

Die Entstehung von BDD wird allgemein Dan North zugeschrieben, der dieses Konzept im Jahr 2006 in einem Artikel[14] vorstellte. North suchte nach Wegen, um die Effektivität von Test-Driven Development zu verbessern, insbesondere im Hinblick auf die Kommunikation und das Verständnis zwischen den Teammitgliedern. Er erkannte, dass die Hauptprobleme bei der Softwareentwicklung oft weniger in der technischen Umsetzung als in Missverständnissen über die gewünschten Ergebnisse liegen. Daher entwickelte er BDD als Ansatz, der die Bedeutung klarer Kommunikation betont, indem er Szenarien in einer natürlichen Sprache beschreibt, die von allen Beteiligten verstanden wird.

Folgendermaßen könnte ein Testfall aussehen, der eine erfolgreiche Anmeldung auf einer Webseite beschreibt, geschrieben in einem Given-When-Then Syntax.

```
Feature: Benutzeranmeldung
  Scenario: Erfolgreiche Anmeldung mit gültigen Anmeldedaten
    Given ich befinde mich auf der Anmeldeseite
    When ich gebe einen gültigen Benutzernamen und ein gültiges Passwort ein
    And ich klicke auf den Anmeldebutton
    Then sollte ich zum Dashboard weitergeleitet werden
    And ich sollte „Willkommen“ sehen
```

Abbildung 5 Einfaches Code-Beispiel für ein Given-When-Then Szenario

Ein zentraler Aspekt von BDD selbst ist das Behavior-Driven-Testing, ein Ansatz, der aus der Praxis von BDD entstanden ist. Es integriert sich in traditionelle Softwareentwicklungsprozesse, indem es die Kommunikation zwischen Teammitgliedern und Stakeholdern verbessert und sicherstellt, dass die Software den Geschäftsanforderungen entspricht [74]. In der Anforderungsanalysephase werden Anforderungen in Form von Given-When-Then Szenarien dokumentiert, wie das Zurücksetzen eines Passwortes:

```
Feature: Passwort-Zurücksetzung
  Scenario: Erfolgreiche Zurücksetzung des Passworts mit gültigen Anmeldedaten
    Given ich befinde mich auf der Seite zur Passwort-Zurücksetzung
    When ich gebe meine registrierte E-Mail-Adresse ein
    And ich klicke auf den Link zur Passwort-Zurücksetzung in der E-Mail
```

[14] https://dannorth.net/introducing-bdd/

```
And ich gebe ein neues, gültiges Passwort ein
And ich bestätige das neue Passwort
Then sollte ich eine Bestätigungsmeldung sehen
And ich sollte mich mit dem neuen Passwort anmelden können
```

Abbildung 6 Komplexeres Code-Beispiel für ein Given-When-Then Szenario

Diese Szenarien dienen als Grundlage für das Design, die Implementierung und das Testen.

Die Implementierung von Behavior-Driven-Development und Behavior-Driven-Testing basiert auf dem Einsatz spezifischer Werkzeuge und Frameworks, die Verhaltensszenarien definieren und entsprechende Tests ausführen können. Ein zentrales Element ist das 'Given-When-Then'-Format, das eine klare und strukturierte Methode bietet, um Anforderungen und Verhalten zu beschreiben. Tools wie Cucumber[15] und SpecFlow[16] nutzen dieses Format, um Teams zu ermöglichen, Szenarien in einer für alle verständlichen Weise darzustellen. Diese Vorgehensweise fördert die effektive Zusammenarbeit zwischen allen Beteiligten und unterstützt ein gemeinsames Verständnis der Ziele und Anforderungen der Software [73].
Die Vorteile von BDD sind eindeutig: eine verbesserte Verständlichkeit der Projektanforderungen, eine höhere Qualität der Softwareprodukte und eine effektivere Zusammenarbeit innerhalb des Teams und mit den Stakeholdern. Zusätzlich können die Abläufe bereits sehr früh aus den Anforderungen abgeleitet werden, was sicherstellt, dass sie für alle Beteiligten von Anfang an klar und verfügbar sind. Dies erleichtert nicht nur die Entwicklung, sondern auch die Abstimmung zwischen den verschiedenen Teammitgliedern und Stakeholdern [73].

Durch das Verständnis und die Anwendung von Behavior-Driven-Development und Testing können Entwicklerteams Software erstellen, die nicht nur technischen Anforderungen genügt, sondern auch in enger Abstimmung mit den Wünschen und Bedürfnissen der Endnutzer entwickelt wird. BDD fördert eine Kultur der Offenheit, des Dialogs und der kontinuierlichen Verbesserung, die es ermöglicht, hochwertige Softwareprodukte effizient und effektiv zu liefern [75].

3.4 Low-Code/No-Code Plattformen

Um das Konzept von Low-Code-Plattformen genauer zu definieren, ist es notwendig, sie von No-Code-Plattformen zu unterscheiden. No-Code-Plattformen sollen eine Programmierung komplett ohne Code ermöglichen. Dies kann durch Methoden wie Drag-and-Drop oder Capture-and-Replay Mechanismen erreicht werden [9].

In der Qualitätssicherung kann der Anmeldevorgang mit einer No-Code-Plattform einfach getestet werden, indem man den Vorgang manuell durchführt und aufzeichnet, um daraus einen Test zu generieren. Diese Methode ist zwar schnell und intuitiv, stößt jedoch auf Herausforderungen, insbesondere bei hardwareabhängigen Prozessen. Beispielsweise könnte die Steuerung eines Roboterarms nicht im Lernmodus aufgezeichnet

[15] https://cucumber.io
[16] https://specflow.org

werden, da dieser Modus nur zur manuellen Steuerung dient. Stattdessen sollte die Ansteuerung des Roboterarms direkt über das Capture/Replay-Verfahren erfolgen oder im Anschluss in den Testablauf integriert werden. Zudem könnten zusätzliche Sensoren wie Kameras oder haptische Systeme erforderlich sein, um sicherzustellen, dass der Roboter die Bewegungen präzise und wie vorgesehen ausführt [76].

Tests müssen idempotent sein, um reproduzierbare Ergebnisse sicherzustellen und unerwünschte Seiteneffekte bei mehrfacher Ausführung zu vermeiden. Generierte Tests können jedoch fragil sein und bei Änderungen in der Umgebung oder den Abläufen fehlschlagen, was regelmäßige Anpassungen erfordert, um ihre Stabilität sicherzustellen. Zudem kann die parallele Ausführung von Tests zu unerwarteten Ergebnissen führen, insbesondere wenn mehrere Tests gleichzeitig auf dieselben Ressourcen zugreifen. Automatisierte Testsysteme, egal ob für Software oder Hardware, bieten eine einfache Möglichkeit zur Testdurchführung, erfordern aber sorgfältige Planung, um Wiederholbarkeit, Robustheit und Stabilität zu gewährleisten [77], [78].

Im Gegensatz zu No-Code-Plattformen, die vollständig ohne Programmierung auskommen, nutzen Low-Code-Plattformen eine minimale Menge an Code [9]. Zwar ist auch hier Capture-and-Replay möglich, jedoch führt dies zur Generierung von strukturiertem und teilweise lesbarem Code und nicht nur einer visuellen Repräsentation davon. Der Hauptvorteil von Low-Code gegenüber No-Code besteht in der größeren Flexibilität und Anpassungsfähigkeit, die es ermöglicht, komplexere Testfälle durch die Einbindung benutzerdefinierter Code-Snippets zu erstellen und zu erweitern [79]. Insbesondere bei der Steuerung von Robotern zeigt sich dieser Vorteil deutlich: Low-Code-Plattformen erlauben eine differenzierte Steuerung durch die Möglichkeit, manuelle und lernbasierte Steuerungsbefehle nachträglich in die Testskripte zu integrieren. Dies eröffnet nicht nur eine präzisere Anpassung an spezifische Testszenarien, sondern bietet auch eine Grundlage, die Lernschritte der Robotersteuerung zu erfassen und für wiederkehrende Abläufe zu optimieren. Die Plattform kann dadurch Roboteraktionen effizient reproduzieren und durch das Lernen aus vorherigen Interaktionen weiter verbessern.

Allerdings bringen Low-Code-Plattformen auch einige Nachteile mit sich. Ein häufiger Kritikpunkt ist die Abhängigkeit von proprietären Sprachen, was die Portabilität der Anwendungen einschränkt und einen Vendor-Lock-In-Effekt verursacht, durch den Unternehmen an die Technologie eines spezifischen Anbieters gebunden sind [80]. Dies kann insbesondere problematisch sein, wenn Teile des Codes in proprietären oder niedrigeren Programmiersprachen generiert und somit für Low-Code-Entwickler:innen schwer zugänglich sind. Dadurch wird die Flexibilität der Entwicklung eingeschränkt, und die Nachvollziehbarkeit und Anpassung solcher automatisch generierten Teile ist häufig erschwert. Zusätzlich leiden einige Plattformen unter eingeschränkter Skalierbarkeit, und viele Entwickler:innen sind nicht ausreichend geschult, um die Vorteile und Funktionen der Low-Code-Umgebung, insbesondere in Bezug auf Robotersteuerung und das Einbinden lernbasierter Prozesse, voll auszuschöpfen [81], [82].

Low-Code- und No-Code-Plattformen zielen beide darauf ab, die Softwareentwicklung schneller und leichter zugänglich zu machen, aber sie tun dies auf unterschiedliche Weise und sprechen unterschiedliche Zielgruppen an [78].

Low-Code-Plattformen sind in der Regel für Entwickler:innen und technisch versierte Personen beziehungsweise Fachexpert:innen konzipiert. Sie bieten eine hohe Flexibilität und sind in der Lage, komplexe und angepasste Anwendungen zu unterstützen. Dies wird durch die Möglichkeit erreicht, Code hinzuzufügen oder zu ändern, was eine moderate Lernkurve und technisches Know-how für Wartung und Support erfordert. Die Entwicklungsgeschwindigkeit ist zwar höher als bei der herkömmlichen Codierung, aber langsamer als bei No-Code-Plattformen [78].

Andererseits sind No-Code-Plattformen ideal für Personen ohne Programmierkenntnisse. Sie ermöglichen eine sehr schnelle Entwicklung durch Drag-and-Drop-Design und sind ideal für einfache bis mittelkomplexe Anwendungen. Die Flexibilität ist begrenzt, da feingranulare Codeänderungen nicht wirklich möglich sind, was eine flache Lernkurve zur Folge hat. Wartung und Support werden in der Regel vom Plattformanbieter übernommen, was sie zu einer benutzerfreundlichen Option für technisch nicht versierte Anwender:innen macht [78].

Die Wahl zwischen Low-Code- und No-Code-Plattformen hängt von den spezifischen Bedürfnissen und Fähigkeiten des Nutzers ab. Beide haben ihre Stärken und können in den richtigen Kontexten sehr effektiv sein. Eine Zusammenfassung der Vor- und Nachteile sind in Tabelle 1 zusammengefasst [78]:

Merkmal	**Low-Code Plattformen**	**No-Code Plattformen**
Zielgruppe	Entwickler:innen und technisch versierte Personen	Personen ohne Programmierkenntnisse
Komplexität der Anwendungen	Geeignet für komplexe und maßgeschneiderte Anwendungen	Ideal für einfache bis mittelkomplexe Anwendungen
Flexibilität	Hohe Flexibilität durch die Möglichkeit, Code hinzuzufügen oder zu ändern	Begrenzte Flexibilität, da keine Code-Änderungen möglich sind
Entwicklungsgeschwindigkeit	Schneller als traditionelle Codierung, aber langsamer als No-Code	Sehr schnelle Entwicklung durch Drag-and-Drop-Design
Lernkurve	Moderate Lernkurve, da einige Programmierkenntnisse hilfreich sind	Flache Lernkurve, da keine Programmierkenntnisse erforderlich sind
Wartung und Support	Erfordert technisches Know-how für Wartung und Support	Wartung und Support sind in der Regel durch den Plattformanbieter abgedeckt

Tabelle 1 Unterschiede Low-Code Plattformen & No-Code Plattformen nach [78]

Laut [70] und [83] erleichtern Low-Code-Plattformen die Entwicklung von Softwareanwendungen, um aktuelle Unternehmensanforderungen zu unterstützen und eine Digitalisierung zu fördern. Besonders nützlich sind diese Plattformen in der Fertigungsindustrie, wo sie es Fachexpert:innen ohne Programmierkenntnissen, Anwendungen und IoT-

Geräten ermöglichen, nahtlos in einer vernetzten Umgebung zu kommunizieren und zu interagieren.

Bei Low-Code-Plattformen geht es nicht nur darum, die Menge des geschriebenen Codes zu reduzieren; Sie bieten außerdem ein hohes Maß an Abstraktion, das es Anwender:innen ermöglicht, sich auf die Logik und den Ablauf statt auf die Implementierungsdetails zu konzentrieren. Dies macht sie zu einem hervorragenden Werkzeug für die schnelle Anwendungsentwicklung, da sie eine schnelle Prototypenerstellung und Iteration ermöglichen [84].

Es ist jedoch wichtig zu beachten, dass Low-Code-Plattformen zwar den Entwicklungsprozess erheblich beschleunigen und ihn für Nicht-Programmierer zugänglicher machen können, es sich jedoch nicht um eine Einheitslösung handelt. Für spezifische Anwendungen ist es notwendig, die Plattform ebenfalls spezifisch anzupassen, um die individuellen Anforderungen und Ziele des Projekts zu erfüllen. Dies liegt daran, dass jede Anwendung einzigartige Anforderungen und Herausforderungen mit sich bringt, die oft über die Standardfunktionen einer Low-Code-Plattform hinausgehen. Laut [85] fehlen oft definierte Richtlinien für die Anforderungserhebung in solchen spezifischen Projekten, was zusätzliche Anpassungen und Erweiterungen erforderlich macht. Diese Anpassungen umfassen die Integration spezifischer Geschäftslogiken, maßgeschneiderter Benutzeroberflächen und spezialisierter Datenverarbeitungsprozesse, die nicht durch standardisierte Lösungen abgedeckt werden können [85].

3.5 Limitationen von Low-Code/No-Code Plattformen

Low-Code-Plattformen bieten zwar eine Vielzahl von Vorteilen wie eine schnelle Entwicklung und einen geringeren Bedarf an umfassenden Programmierkenntnissen, sind jedoch nicht ohne Nachteile.

Ein Artikel der Freien Universität Berlin [73] liefert eine umfassende Analyse der Einschränkungen, die bei der Entscheidung für einen Low-Code-Ansatz unbedingt zu berücksichtigen sind:

- Begrenzte Anpassbarkeit
- Anbieterabhängigkeit
- Effizienz des generierten Codes
- Zeitersparnis und Entwicklungszeit
- Sicherheitsbedenken
- Kosten

Ein Hauptnachteil von Low-Code- und No-Code-Plattformen ist ihre eingeschränkte Anpassbarkeit. Diese Plattformen zeichnen sich zwar dadurch aus, dass sie die Entwicklung von Standardanwendungen beschleunigen können jedoch bei komplexen Anforderungen, die ein hohes Maß an Anpassung erfordern, in Probleme geraten. Die vorgefertigten Komponenten und Vorlagen solcher Plattformen können zum Hindernis werden,

wenn komplexe Funktionen oder individuelle Anpassungen benötigt werden, die außerhalb des vorgegebenen Rahmens liegen [79].

Die Integration von Hardware in No-Code- und Low-Code-Anwendungen bringt besondere Herausforderungen mit sich, da diese Plattformen im Vergleich zur traditionellen Programmierung weniger Flexibilität bieten [73]. Bei der traditionellen Programmierung kann der Entwickler:innen den Code direkt an die Hardware anpassen, wodurch eine nahtlose Kommunikation zwischen verschiedenen Komponenten wie Motoren und Sensoren ermöglicht wird [86]. In No-Code- und Low-Code-Umgebungen hingegen sind die eingesetzten domänenspezifischen Sprachen oft auf bestimmte Systeme (z.B. Webanwendungen) optimiert, was bedeutet, dass zusätzliche Anpassungen oder die Implementierung von Schnittstellen erforderlich sind, um verschiedene Hardwaremodule zu verbinden. Dies führt zu einem erhöhten Aufwand und höheren Kosten, da die Plattformen nicht von Haus aus auf eine breite Palette an Hardwarekomponenten ausgelegt sind [87].

Traditionell programmierte Systeme erlauben eine feingranulare Steuerung, beispielsweise durch die direkte Konfiguration von Sensoren und Aktoren in Echtzeit [86]. In Low-Code- und No-Code-Umgebungen fehlt diese Möglichkeit oft, da die Plattformen keine direkten Eingriffe in die Hardwaresteuerung erlauben. Bei der Integration von komplexen Komponenten, die auf eine präzise Synchronisation angewiesen sind, um genaue Daten zu liefern, können daher Synchronisationsprobleme auftreten. Diese entstehen, wenn unterschiedliche Hardwaremodule nicht zeitgleich oder koordiniert arbeiten. Solche Probleme führen zu Ungenauigkeiten in der Robotiksteuerung und können die Funktionalität erheblich beeinträchtigen [88]. Abhilfe schaffen hier spezialisierte Frameworks oder Middleware, die automatisch eine Synchronisation zwischen den Hardwarekomponenten gewährleisten, was jedoch wiederum mit zusätzlichem Entwicklungsaufwand und eventuellen Kosten verbunden ist [88].

Die Anbieterabhängigkeit ist ein weiteres wichtiges Problem. Die Einführung so einer Plattform kann zu einem hohen Maß an Abhängigkeit vom Plattformanbieter führen. Änderungen der Geschäftsbedingungen des Anbieters, Preiserhöhungen oder gar die Einstellung der Plattform können erhebliche Auswirkungen auf die darauf entwickelten Projekte haben [80].

Low-Code-Plattformen können bei der Umsetzung komplexer Funktionen zeitaufwendiger sein, weil oft nur wenige vorgefertigte Lösungen verfügbar sind. Das bedeutet, dass Entwickler:innen mehr Zeit mit der Recherche und Anpassung verbringen müssen. Dies ist darauf zurückzuführen, dass die Plattformen vor allem für standardisierte Aufgaben optimiert sind und weniger Flexibilität bei spezifischen, komplexen Anforderungen bieten [73].

Ein weiterer Nachteil ist die Leistung des automatisch generierten Codes. Dieser Code kann langsamer und weniger effizient sein als von Hand geschriebener Code, was zu schlechterer Gesamtleistung führen kann. Automatisiert generierter Code ist oft weniger optimiert und kann unnötigen Ballast enthalten, was die Ausführungsgeschwindigkeit beeinträchtigt. Zudem kann die Wartung des Codes schwieriger sein, da automatisch generierter Code häufig weniger übersichtlich und schwer verständlich ist. Dies führt

dazu, dass Entwickler:innen mehr Zeit und Aufwand investieren müssen, um den Code zu pflegen und anzupassen [90].

Bei Low-Code- und No-Code-Plattformen bestehen Sicherheitsbedenken, da diese Plattformen den Großteil des Codes automatisch generieren. Dadurch haben Entwickler:innen weniger Einfluss auf die sicherheitsrelevanten Aspekte der Anwendung, was die Sicherheit beeinträchtigen und die Anwendung anfällig für Schwachstellen machen kann [91].

Ein weiterer Nachteil sind die potenziell hohen Kosten für die Nutzung von Low-Code- und No-Code-Plattformen, vor allem für größere Unternehmen oder Anwendungen mit einer großen Anzahl von Nutzern. Zwar können diese Plattformen die Entwicklungszeit deutlich reduzieren, aber die Lizenz- und Nutzungsgebühren können sich summieren und langfristig zu einer kostspieligen Lösung werden [92].

Zusammenfassend lässt sich sagen, dass Low-Code- und No-Code-Plattformen viele Vorteile bieten, ihre Einschränkungen und Nachteile jedoch sorgfältig abgewogen werden müssen. Diese Nachteile können je nach Plattform und Anwendungsfall unterschiedlich ausfallen, was die Bedeutung einer gründlichen Evaluierung vor der Einführung unterstreicht.

3.6 Domain-Specific-Languages

Domain-Specific-Languages (DSLs) sind spezialisierte Programmiersprachen, die für eine bestimmte Anwendungsdomäne entwickelt wurden. Sie bieten ein hohes Maß an Abstraktion und Ausdruckskraft und sind damit ein leistungsstarkes Werkzeug im Bereich der Softwareentwicklung. Durch die Verwendung von DSLs können Entwickler:innen Code schreiben, der besser lesbar und wartbar ist, was zu einer höheren Produktivität und weniger Fehlern führen kann [93].

DSLs fördern die Zusammenarbeit zwischen Entwickler:innen und Domänenexpert:innen, da sie eine gemeinsame Sprache für die Beschreibung und Analyse von Problemen bieten. Dies verbessert das Verständnis und die Präzision bei der Umsetzung von Anforderungen [94]. Dazu gehört die Erstellung von Modellierungstools, die es Fachexpert:innen ermöglichen, ihre Anforderungen zu spezifizieren. Diese Tools können dann zum Generieren einer Implementierung verwendet werden und so die Lücke zwischen Domänenentwickler:innen und Softwareentwickler:innen schließen.

DSLs sind besonders dann von Vorteil, wenn Sie über ein Softwaresystem verfügen, das Sie über einen längeren Zeitraum nutzen und warten möchten. Sie ermöglichen es Fachanwender:innen, spezifische fachliche Abläufe und Konzepte klar und präzise darzustellen, was sie zu einer wertvollen Investition macht. Durch die Spezialisierung auf bestimmte Aufgabenbereiche können DSLs die Entwicklung von Prototypen und die Implementierung von Lösungen erheblich beschleunigen. Dies ist besonders nützlich in der frühen Phase der Softwareentwicklung, wo schnelle Iterationen erforderlich sind [95].

Allerdings sind die Entwicklung und Nutzung von DSLs nicht ohne Herausforderungen. Eine systematische Übersichtsstudie zu DSLs, wie sie in [93] vorgestellt wird, zeigt, dass sich die DSL-Community mehr auf die Entwicklung neuer Techniken und Methoden konzentriert, als die Integration von DSLs mit anderer Softwaretechnik zu untersuchen Prozesse oder die Messung der Wirksamkeit von DSL-Ansätzen. Die Studie weist auch auf einen deutlichen Mangel an Evaluierungsforschung in diesem Bereich hin. Die Studie zeigt, dass die Entwicklung von Domain-Specific-Languages (DSLs) zwar ein aktives Forschungsfeld ist, jedoch gibt es mehrere Herausforderungen. Es fehlt häufig an Forschung zur Integration von DSLs in bestehende Softwareentwicklungsprozesse und an systematischen Evaluierungen ihrer Wirksamkeit in der Praxis. Die meisten Tools unterstützen nur spezifische Aspekte der DSL-Entwicklung, was die Konfiguration von Toolchains und die Auswahl geeigneter Werkzeuge erschwert. Eine weitere Herausforderung liegt in der begrenzten Unterstützung für bidirektionale Notationen (z. B. zwischen grafischen und textuellen Modellen) sowie in der mangelnden Unterstützung für Metamodell-Migration und Modell-Simulation. Insgesamt gibt es also zwar viele Werkzeuge, aber die nahtlose Integration und der praktische Einsatz von DSLs in größeren Entwicklungsprozessen sind bisher nur unzureichend erforscht [93].

In den verschiedenen Entwicklungsphasen einer DSL ist besondere Aufmerksamkeit auf Domänenanalyse, Validierung und Wartung erforderlich. Die Domänenanalyse hilft, die spezifischen Anforderungen und Grenzen der Domäne zu verstehen und zu modellieren [96]. Die Validierung stellt sicher, dass die DSL syntaktisch, semantisch und Bezug auf die spezifischen Anwendungsanforderungen korrekt ist, und identifiziert potenzielle Designfehler [97], [98]. Die kontinuierliche Wartung und Weiterentwicklung der DSL sind entscheidend, um ihre langfristige Nützlichkeit und Anpassungsfähigkeit an neue Anforderungen zu gewährleisten [99].

Zusammenfassend lässt sich sagen, dass DSLs eine entscheidende Rolle in der modernen Softwareentwicklung spielen und ein leistungsstarkes Werkzeug zur Lösung domänenspezifischer Probleme darstellen [100] [101].

3.7 Keyword-Driven-Testing

Keyword-Driven-Testing (KDT) ist ein Ansatz der Testautomatisierung, bei dem vordefinierte Schlüsselwörter verwendet werden, um Testaktionen zu steuern. Diese Schlüsselwörter stehen für bestimmte Testaktionen wie das Klicken auf Buttons oder das Eingeben von Text und werden zusammen mit den zugehörigen Testschritten dokumentiert. Diese Methodik zielt darauf ab, den Testprozess effizienter zu gestalten, indem die Testlogik von der Implementierung getrennt wird. Durch die Verwendung von Schlüsselwörtern können Tests von Anwender:innen erstellt werden, die keine umfassenden Programmierkenntnisse besitzen, was besonders in großen Entwicklungsteams von Vorteil ist, wo nicht jeder/jede technische Expertise besitzt [102].

Ein wesentlicher Vorteil von Keyword-Driven-Testing liegt in der Wiederverwendbarkeit der Testfälle. Da die Testlogik in verständliche Schlüsselwörter abstrahiert wird, können diese Keywords über mehrere Tests hinweg wiederverwendet werden, was den

Wartungsaufwand erheblich reduziert und den Testprozess beschleunigt. Teständerungen müssen nur einmal zentral vorgenommen werden, was das Management von Testfällen erleichtert und die Wahrscheinlichkeit von Fehlern minimiert. Ein weiteres Plus von KDT ist die Zugänglichkeit für nicht-technische Stakeholder, da die Testfälle in einer leicht verständlichen Form vorliegen und somit von verschiedenen Teammitgliedern, einschließlich Business-Analyst:innen und Produktmanager:innen, überprüft und bearbeitet werden können. Dies fördert die Zusammenarbeit und ermöglicht eine bessere Kommunikation zwischen den technischen und nicht-technischen Bereichen eines Projekts [103].

Ein Beispiel für eine weitverbreitete Anwendung von Keyword-Driven Testing ist das Robot Framework[17]. Dieses Framework, das ebenfalls auf der Idee basiert, Tests mit Hilfe von Schlüsselwörtern zu steuern, hat sich zu einem beliebten Werkzeug in der Testautomatisierung entwickelt. Insbesondere in Kombination mit Tools wie Selenium[18] bietet das Robot Framework eine leistungsstarke Möglichkeit, um Tests zu automatisieren, ohne dass tiefgehende Programmierkenntnisse erforderlich sind. Ein großer Vorteil des Robot Frameworks ist seine Integration in Continuous Integration (CI)-Tools, was eine kontinuierliche Testausführung ermöglicht. Dadurch können regelmäßig Regressionstests durchgeführt werden, was die Qualität der Software kontinuierlich sicherstellt und es den Teams ermöglicht, schneller auf Änderungen zu reagieren [104]. Abbildung 7 zeigt ein Beispiel für den Einsatz von Keyword-Driven-Testing. In diesem Fall wird ein Testfall mit dem Robot Framework dargestellt, der eine Suchanfrage auf einer Suchmaschine automatisiert überprüft.

```
*** Settings ***
Library    SeleniumLibrary

*** Variables ***
${search_term}    Robot Framework
${expected_result}    Robot Framework

*** Test Cases ***
Automate Google Search
    Open Browser to Google
    Input Search Term
    Click Search Button
    Verify Search Results

*** Keywords ***
Open Browser to Google
    Open Browser    https://www.google.com    Chrome
    Maximize Browser Window

Input Search Term
    Input Text    name=q    ${search_term}

Click Search Button
    Click Button    name=btnK

Verify Search Results
    Page Should Contain    ${expected_result}
```

Abbildung 7 Beispiel für einen KDT-Testfall der mit dem Robotframework umgesetzt wurde

[17] https://robotframework.org
[18] https://www.selenium.dev

Keyword-Driven-Testing steht auch in engem Zusammenhang mit Behavior-Driven Development (BDD, siehe 3.3). Während BDD sich auf natürlichsprachliche Testfälle (z.B. in der „Given-When-Then“-Struktur wie in Kapitel 3.3) fokussiert, verfolgt KDT einen ähnlichen Ansatz, indem es Tests in Form von Schlüsselwörtern beschreibt. Beide Ansätze zielen darauf ab, die Testautomatisierung verständlicher und zugänglicher zu machen, besonders für nicht-technische Teammitglieder. In vielen Fällen können KDT und BDD kombiniert werden, um eine umfassende Teststrategie zu entwickeln, bei der sowohl die funktionalen Anforderungen als auch die technischen Details in einer für alle Beteiligten verständlichen Form dokumentiert werden [105].

Keyword-Driven-Testing ist ein wichtiger Bestandteil moderner Testautomatisierungsstrategien. Es vereinfacht den Testprozess, fördert die Zusammenarbeit zwischen verschiedenen Teammitgliedern und erleichtert die Wartung von Tests. In Verbindung mit BDD zeigt sich, dass KDT ein flexibles und anpassungsfähiges Werkzeug ist, das sowohl in technisch komplexen als auch in einfacheren Entwicklungsumgebungen wertvolle Dienste leistet [104], [106].

4 Aktueller Stand der Technik

In diesem Abschnitt wird ein Überblick über bereits bestehende Ansätze präsentiert, die sich mit robotergestützter Testautomatisierung, No-Code- und Low-Code-Plattformen sowie deren Kombination auseinandersetzen.

4.1 Fallstudien zu Low-Code- und No-Code-Plattformen

In den letzten Jahren haben Low-Code- und No-Code-Plattformen (LCNC) einen signifikanten Einfluss auf die Art und Weise, wie Software entwickelt wird. Der Markt für derartige Entwicklungsplattformen wird von 2021 bis 2026 laut einer Marktstudie von Technavio [107] voraussichtlich um 35,73 Milliarden USD wachsen, was einer jährlichen Wachstumsrate von 27,96 % entspricht. Treiber für das Marktwachstum sind unter anderem die Preisstrategien der Anbieter sowie die zunehmende digitale Transformation in der IT-Branche [107].

Der Bericht von N. Prinz et al. [108] betont die wachsende Bedeutung von Low-Code-Entwicklungsplattformen aufgrund ihrer Fähigkeit, die Softwareentwicklung zu beschleunigen und Kosten zu senken. Er analysiert 32 wissenschaftliche Publikationen und zeigt, dass der Großteil der Forschung sich auf technische Aspekte konzentriert und es nur wenige Studien zu sozialen Faktoren gibt. Es ist ebenso wichtig, wie gut Teams die neuen Plattformen akzeptieren und integrieren, sowie die Schulung und Unterstützung der Mitarbeiter, um die neuen Tools effektiv zu nutzen [108].

Die Analyse der Diskussionsinhalte von Entwickler:innen in der Studie [109] von Md. Abdullah Al Alamin et al. verdeutlicht, dass LCNC-Plattformen durch ihre Benutzerfreundlichkeit den Softwareentwicklungsprozess zugänglicher gestalten. Allerdings sehen Entwickler:innen auch Herausforderungen, wie etwa die begrenzte Anpassungsfähigkeit und die Schwierigkeiten bei der Integration in bestehende IT-Infrastrukturen. Die Ergebnisse zeigen, dass die häufigsten Herausforderungen die begrenzte Flexibilität und die mangelnde Skalierbarkeit von Low-Code-Plattformen sind, sowie die Notwendigkeit, sich an die vorgegebenen Strukturen und Funktionen der Plattform anzupassen. Es wurde auch festgestellt, dass Entwickler:innen Schwierigkeiten bei der Fehlerbehebung und dem Debugging von Low-Code-Anwendungen haben. Einige Entwickler:innen berichteten auch, dass es schwierig ist, die Leistung und die Sicherheit von Low-Code-Anwendungen zu gewährleisten [109]. Die effektive Nutzung von LCNC-Plattformen erfordert umfassende Schulungen und ausführliche Dokumentation, um ihr volles Potential auszuschöpfen. Insgesamt zeigt das Paper, dass Low-Code-Plattformen eine schnelle und einfache Möglichkeit bieten, Anwendungen zu erstellen, aber dass es auch Herausforderungen und Einschränkungen gibt, die berücksichtigt werden müssen. Trotz dieser Barrieren werden die Vorteile in Bezug auf Geschwindigkeit und Effizienz

in der Entwicklung von vielen Entwickler:innen als entscheidend für den zukünftigen Erfolg von Low-Code-Plattformen angesehen.

K. Rokis und M. Kirikova untersuchen in [110] ähnlich wie in [109] die Herausforderungen, die mit der Nutzung von Low-Code/No-Code-Entwicklungsplattformen verbunden sind. Der Literaturüberblick zeigt, dass trotz der Vorteile wie schnellere Entwicklung und geringere Kosten auch Herausforderungen mit sich bringen. Diese umfassen technische Aspekte wie begrenzte Anpassungsmöglichkeiten und Integrationsprobleme sowie soziale Aspekte wie Akzeptanz durch Entwickler:innen und Anwender, Schulungsbedarf und Veränderungsmanagement. Ähnlich wie [108] wird die Notwendigkeit hervorgehoben, sowohl technische als auch soziale Herausforderungen zu adressieren, um die Effektivität und Akzeptanz dieser Plattformen zu maximieren.

Eine weitere Studie zu diesem Thema wurde durch D. Dahlberg in einer explorativen Studie [111] beschrieben, wobei hier konkret bei einem IT-Unternehmen Daten erhoben wurden, wo erst kürzlich ein Umstieg von traditioneller Programmierung auf eine LCNC-Lösung vollzogen wurde. Teilnehmer wurden basierend auf ihrer Erfahrung mit Low-Code-Entwicklung ausgewählt und es wurden Interviews mit sechs LCNC-Entwickler:innen sowie einem Projektleiter geführt. Die Ergebnisse der Studie spiegeln die zuvor diskutierten Erkenntnisse wider.

Positive Erfahrungen:

- Erhöhte Produktivität
- Verbesserte Kundenbeziehungen
- Konzentration auf das eigentliche Ziel
- Gemeinsames Verständnis unter den Entwickler:innen
- Schnelle Erlernbarkeit

Negative Erfahrungen:

- Eingeschränkte Arbeitsweise
- Eingeschränkte Freiheit und Kreativität
- Unzureichende Dokumentation
- Mangelnde Übersicht
- Schwache und unsichere Teamarbeit

LCNC-Plattformen eignen sich laut der Studie von D. Dahlberg [111] besonders für Projekte mit überschaubaren Komponenten, wie beispielsweise einfache Webseiten, die vollständig im Browser laufen. Bei komplexen Systemen mit mehreren Komponenten wie mobile Apps und Hardware gibt es allerdings Einschränkungen bei der Flexibilität und Skalierbarkeit, die berücksichtigt werden müssen.

Die Studie „Low-Code Platforms and Languages: The Future of Software Development“ von G. Juhás et al. [112] beleuchtet die Rolle von Low-Code-Plattformen in der Softwareentwicklung. Die Autoren argumentieren, dass LCNC-Plattformen die Lücke zwischen IT-Expert:innen und Nicht-Programmierer:innen überbrücken. Durch die Bereitstellung einer einfach zu bedienenden visuellen Entwicklungsumgebung ermöglichen sie es Anwender:innen, Anwendungen schneller und mit weniger Fehlern zu ent-

wickeln. Die Studie prognostiziert, dass zukünftige Generationen von Low-Code-Plattformen noch leistungsfähiger werden und es Unternehmen ermöglichen werden, agil zu bleiben, da die Entwicklung und Anpassung von Softwareprojekten beschleunigt wird. Ein zentraler Punkt dieser Studie ist die Untersuchung der Effizienz, die Low-Code-Plattformen in den Entwicklungsprozess einbringen. Im Gegensatz zur traditionellen Softwareentwicklung ermöglichen Low-Code-Plattformen die Entwicklung funktionaler Prototypen in signifikant weniger Zeit [112]. Die Autoren argumentieren, dass dieser Paradigmenwechsel besonders in einem geschäftlichen Umfeld mit hohen Anforderungen an die Geschwindigkeit von Entwicklungszyklen von entscheidender Bedeutung ist.

In der Studie „Challenges of Low-Code/No-Code Software Development: A Literature Review“ [110] von K. Rokis und M. Kirikova werden die verschiedenen Herausforderungen von LCNC-Plattformen umfassend analysiert. Die Autoren konzentrieren sich auf technische Hürden wie die eingeschränkte Anpassbarkeit, die Schwierigkeit der Integration in bestehende IT-Systeme und die Unsicherheit in Bezug auf die langfristige Wartbarkeit der erstellten Anwendungen. Die Autoren heben hervor, dass trotz der offensichtlichen Vorteile wie der schnellen Anwendungsentwicklung und der Zugänglichkeit für Nicht-Programmierer:innen zahlreiche technische Herausforderungen bestehen. Insbesondere bei komplexeren Anforderungen, die über einfache Geschäftsprozesse hinausgehen, stoßen Low-Code/No-Code-Plattformen an ihre Grenzen. Die Anpassung bestehender Lösungen oder die Integration mit anderen unternehmensweiten Systemen erfordert oft tiefere Programmierkenntnisse, was den ursprünglichen Zweck der Plattformen infrage stellt. Die Studie zeigt, dass LCNC-Plattformen zwar vielversprechend sind, aber eine durchdachte Implementierung und Planung erfordern, um ihre vollen Potenziale auszuschöpfen [110].

Die Studie [113] von Dražen M. Marinković und Z. Avramović hebt hervor, dass No-Code-Plattformen nicht nur durch die Abstraktion von Programmierkenntnissen, sondern insbesondere durch ihre Fähigkeit, die Kreativität und Problemlösungsfähigkeiten von Anwender:innen zu fördern, einzigartig sind. Im Gegensatz zu traditionellen Entwicklungsumgebungen, die oft technische Barrieren aufbauen, ermöglicht diese visuelle Programmierung eine direktere und intuitivere Umsetzung von Ideen. Die Autoren argumentieren, dass dies nicht nur die Effizienz erhöht, sondern auch zu innovativeren und benutzerzentrierteren Softwarelösungen führt, da Entwickler:innen ihre volle Aufmerksamkeit auf das Design und die Funktionalität richten können, ohne durch technische Hürden eingeschränkt zu sein.

Im Rahmen des NexusBRaNT-Experiments [114] wurde ein direkter Vergleich zwischen traditioneller und Low-Code-Entwicklung durchgeführt, um die Effizienz beider Ansätze zu bewerten. Die Ergebnisse zeigten, dass die Low-Code-Methode im Vergleich zur traditionellen Programmierung einen enormen Effizienzgewinn bietet, da der Entwicklungsaufwand um mehr als 94 % reduziert werden konnte. Dies wird vor allem durch die Drag-and-Drop-Schnittstellen und die modellgesteuerte Logik ermöglicht, die den Entwicklungsprozess vereinfachen und beschleunigen.

Die bisherigen Studien zu Low-Code- und No-Code-Plattformen (LCNC) betonen die wesentlichen Vorteile dieser Ansätze. So heben Prinz et al. [108] und Rokis & Kirikova

[110] hervor, dass LCNC-Plattformen eine deutliche Beschleunigung der Softwareentwicklung ermöglichen und gleichzeitig die Kosten senken. Durch ihre einfache Bedienbarkeit und visuelle Entwicklungsumgebungen erleichtern sie den Zugang für Nicht-Programmierer:innen, wodurch der Entwicklungsprozess für eine breitere Gruppe von Menschen zugänglich wird. Marinković und Avramović [113] betonen zudem, dass LCNC-Plattformen durch die visuelle Programmierung die Kreativität und Problemlösungsfähigkeiten der Anwender:innenfördern und so innovativere und benutzerzentrierte Lösungen entstehen. Auch das NexusBRaNT-Experiment [114] zeigt, dass durch die Verwendung von Low-Code-Plattformen eine signifikante Effizienzsteigerung möglich ist, da der Entwicklungsaufwand im Vergleich zur traditionellen Programmierung um bis zu 94 % reduziert wird. Diese Vorteile machen LCNC-Plattformen besonders attraktiv für weniger komplexe Projekte, bei denen Geschwindigkeit und Zugänglichkeit entscheidende Faktoren sind.

Gleichzeitig werden jedoch auch einige Herausforderungen identifiziert. Rokis & Kirikova [110] und Dahlberg [111] zeigen auf, dass LCNC-Plattformen bei komplexen Projekten oft an ihre Grenzen stoßen, da sie eine begrenzte Anpassbarkeit und Schwierigkeiten bei der Integration in bestehende IT-Infrastrukturen bieten. Die Studie von Md. Abdullah Al Alamin et al. [109] verdeutlicht außerdem, dass Entwickler:innen häufig mit Problemen bei der Flexibilität, der Skalierbarkeit und der Fehlerbehebung konfrontiert sind.

Die vorliegende Arbeit erweitert diese bestehenden Ansätze, indem sie nicht nur auf die Automatisierung einfacher Entwicklungsprozesse abzielt, sondern eine No-Code-Plattform vorstellt, die deren Nutzer in bestehende Entwicklungsprozesse nachhaltig einbinden soll. Während die bisherigen Studien sich häufig auf spezifische technische oder soziale Herausforderungen konzentrieren, bietet der hier vorgestellte Ansatz eine breitere und flexiblere Lösung, die sowohl einfache als auch komplexere Anwendungsfälle abdeckt.

4.2 Fallstudien zu robotergestützten Testautomatisierung

Verschiedene Studien haben die Vorteile vom Einsatz robotergestützter Testautomatisierung (RTA) gegenüber traditionellen, menschlich durchgeführten Qualitätskontrollen untersucht. Dieses Kapitel fasst mehrere Fallstudien zusammen, die den Einsatz von Robotern in der Qualitätssicherung beleuchten.

In der Studie von L. C. Santos et al. [115] wurde festgestellt, dass die Implementierung von physischen Robotern die Qualitätsprüfung beschleunigte und die Fehlererkennungsrate um 35 % steigerte. Die Studie untersuchte Robotik-Steuerungssysteme, bei denen Software präzise die Bewegungen und Aktionen von Robotern steuert. Vor der Implementierung mussten Entwickler:innen diese Systeme manuell testen, indem sie Roboter in einer simulierten Umgebung steuerten und ihre Reaktionen auf verschiedene Befehle beobachteten. Diese manuelle Erfassung aller möglichen Fehler war problematisch, da die menschliche Aufmerksamkeit und Genauigkeit nachließen. Beispielsweise mussten Bewegungsabläufe, Sensorreaktionen und Sicherheitsmechanismen überprüft werden. Durch den Einsatz von Robotern konnten diese Tests automatisiert und systematischer

durchgeführt werden, indem die Roboter kontinuierlich und unter konstanten Bedingungen getestet wurden, was zu einer gründlicheren Prüfung und einer erheblichen Verbesserung der Softwarequalität führte [115].

In einer 2019 veröffentlichten Fallstudie von IEEE/SICE[19] [116] wurden Roboter zur Qualitätssicherung von webbasierten Anwendungen und mobilen Apps eingesetzt. Vor der Einführung von Robotern zur Qualitätssicherung mussten Entwickler:innen manuell durch den Code gehen, wie es beim White-Box Testen üblich ist (siehe Kapitel 2.6). Ein physischer Roboterarm wurde verwendet, um reale Nutzerszenarien nachzustellen, wie das Drücken von Tasten auf einem Touchscreen oder das Testen von Sensoren in mobilen Geräten. Die Studie zeigte, dass der Einsatz von Robotern zur Qualitätssicherung die Inspektionszeit um 60 % verkürzte und die Fehlerquote um 40 % senkte. Die Automatisierung ermöglichte schnellere und genauere Prüfungen, die zuvor manuell nicht erreicht werden konnten, wodurch die Gesamtqualität der Softwareprojekte erheblich verbessert wurde [116].

In der wissenschaftlichen Arbeit von A. Papavasileiou et al. [117] wurde berichtet, dass Roboter die Fehlererkennungsrate um 50 % in der Qualitätssicherung verbesserten und die Kosten um 30 % senkten. Diese Studie betraf die Fertigungsindustrie, wo manuelle Inspektionen ineffizient und kostspielig waren. Die Prüfer mussten physische Produkte visuell inspizieren und testen, was zu inkonsistenten Ergebnissen führte. Beispielsweise wurden Produkte wie elektronische Bauteile oder mechanische Komponenten überprüft. Durch den Einsatz von Robotern konnten die Inspektionsprozesse automatisiert und die Qualität der Prüfungen erheblich verbessert werden. Die Roboter führten präzise und wiederholbare Tests durch, was die Gesamtqualität der Produkte steigerte und die Produktionskosten senkte [117].

C. Poggi untersuchte in [118] die Automatisierung von Abschluss-Tests in Produktionsprozessen mit einem Roboterarm, der durch das Robot Operating System (ROS) gesteuert wird. Das ROS ist technisch gesehen kein Betriebssystem, sondern ein Software-Framework oder Middleware, das eine integrierte Entwicklungsumgebung für Roboteranwendungen bereitstellt. Es ermöglicht die Kommunikation zwischen verschiedenen Softwarekomponenten, unabhängig davon, ob sie auf demselben oder auf mehreren Computern laufen. Der Roboterarm übernahm Aufgaben wie das Aktivieren von Komponenten und das Messen ihrer Reaktionen, was zuvor manuell durchgeführt wurde. Die Implementierung führte zu einer 50% schnelleren und genaueren Durchführung der Tests, wodurch die Fehlerquote um 30% sank und die Effizienz des Testprozesses deutlich verbessert wurde [118].

In der 2018 über IEEE veröffentlichten Fallstudie [119] untersuchten die Autoren den Einsatz von ROS in der Entwicklung eines autonomen Pick-and-Place-Roboters für industrielle Anwendungen. Der Roboter wurde so konzipiert, dass er Objekte autonom von einem Ort zum anderen bewegen kann. Ein konkretes Beispiel aus der Studie zeigt, wie der Roboter mithilfe von ROS in der Lage war, verschiedene Objekte auf einem Förderband zu erkennen und zu greifen. ROS ermöglichte die Integration der Kamerasensoren, die zur Objekterkennung verwendet wurden, und steuerte den Roboterarm präzise, um die Objekte sicher zu greifen und an einer definierten Stelle abzulegen.

[19] https://sice-si.org

Durch die Nutzung von ROS konnte der Roboter in Echtzeit auf Veränderungen in der Umgebung reagieren und seine Bewegungen entsprechend anpassen. Dies führte laut [119] zu einer signifikanten Verbesserung der Effizienz und Genauigkeit im Produktionsprozess, da manuelle Eingriffe und Fehler reduziert wurden.

Eine Studie von K. A. Khan et al. [120] befasste sich mit der Entwicklung einer Plattform, auf der ein Roboterarm mittels ROS gesteuert wird. Ziel war es, eine präzise Steuerung des Roboterarms in Echtzeit zu ermöglichen. Beispielsweise ist die präzise Montage von Bauteilen in einer Fertigungslinie, wo der Roboterarm kleine Teile genau positioniert und montiert. Das Ergebnis zeigte, dass die Nutzung von ROS eine 40% effizientere und zuverlässigere Steuerung des Roboterarms ermöglicht, was die Genauigkeit und Zuverlässigkeit des gesamten Prozesses erheblich steigerte [120].

Die Fallstudie von G. Walck et al. [121] untersucht die Integration einer Softwarearchitektur basierend auf dem Robot Operating System. Das Projekt zielt darauf ab, Roboterhände zu entwickeln, die auf menschenähnliche Weise Objekte manipulieren können. Hierbei wurden Software- und Hardwarekomponenten eng integriert: ROS verarbeitete Sensordaten und steuerte die Bewegungen der Roboterhand, die mit verschiedenen Sensoren ausgestattet war, um die Position und den Zustand von Objekten zu erkennen. Die Softwarearchitektur ermöglichte Echtzeitanpassungen der Bewegungen, sodass die Roboterhand Objekte sicher greifen und manipulieren konnte. Dies führte zu einer präzisen und zuverlässigen Funktionalität der Roboterhand und zeigte, wie ROS die Entwicklungszeit verkürzen und die Flexibilität des Systems erhöhen kann. Die Studie hebt hervor, wie ROS in IT-Projekten mit Software- und Hardwarekomponenten zur effizienten Bewältigung komplexer Aufgaben beiträgt [121].

N. Demir beschreibt in [60] die Nutzung von ROS zur Weiterentwicklung autonomer mobiler Roboter in der Industrie 4.0. Diese Roboter wurden in einem Lagerhaus eingesetzt, um Waren autonom zu transportieren und dabei in Echtzeit mit dem Lagerverwaltungssystem zu kommunizieren. Dies optimierte die Effizienz der Lagerprozesse erheblich. Die Verwendung von ROS verbesserte die Flexibilität und Skalierbarkeit der Roboterlösungen, was zu einer 30% effizienteren Lagerhaltung und reduzierten Betriebskosten führte [60].

Zusammenfassend lässt sich feststellen, dass der Einsatz von RTA in mehreren Studien zu signifikanten Verbesserungen geführt hat. So zeigt die Studie von L. C. Santos et al. [115], dass durch die Automatisierung mit physischen Robotern die Qualitätsprüfung beschleunigt und die Fehlererkennungsrate um 35 % gesteigert wurde. Ähnlich berichtet die Fallstudie von IEEE/SICE [116] über eine 60%ige Reduktion der Inspektionszeit sowie eine Senkung der Fehlerquote um 40 % durch den Einsatz von Roboterarmen in der Qualitätssicherung mobiler Anwendungen. Die Studie von A. Papavasileiou et al. [117] verdeutlicht, dass Roboter die Fehlererkennungsrate um 50 % verbesserten und gleichzeitig die Kosten um 30 % senkten, insbesondere in der Fertigungsindustrie. Diese und andere Untersuchungen bestätigen, dass RTA den Prüfprozess systematisiert, menschliche Fehler reduziert und gleichzeitig die Effizienz und Präzision deutlich erhöht.

Allerdings bleibt die Implementierung solcher Systeme eine Herausforderung, da sie eine hohe technische Expertise erfordert und oft kostspielige sowie zeitintensive Integrationsprozesse mit sich bringt, wie unter anderem die Studien von C. Poggi [118] und

K. A. Khan et al. [120] zeigen. Insbesondere die Nutzung von ROS setzt umfassende Programmierkenntnisse voraus, um Roboter zuverlässig zu steuern und in bestehende Entwicklungsprozesse zu integrieren. In der vorliegenden Arbeit wird dieses Problem durch die Entwicklung eines Prototyps einer No-Code-Plattform adressiert. Dieses Konzept ermöglicht es Fachexpert:innen, robotergestützte Testautomatisierung ohne Programmierkenntnisse zu nutzen. Die Lösung vereinfacht die Integration von RTA, indem sie Fachexpert:innen direkt in den Entwicklungsprozess einbindet und so eine effizientere und zugänglichere Anwendung von Robotertechnologie in der Qualitätssicherung fördert.

4.3 Fallstudien zu No-Code Plattformen mit robotergestützten Testautomatisierung

Die Studie „A Low-Code Approach for Connected Robots“ [122] befasst sich mit der Entwicklung fortschrittlicher robotischer Systeme, die mit dem Internet der Dinge (IoT) verbunden sind. Solche Systeme bestehen aus vielen verschiedenen Komponenten wie Sensoren, Software und Hardware, was die Entwicklung komplex und zeitaufwendig macht. Um dies zu vereinfachen, schlägt die Studie einen Ansatz vor, der es auch Personen ohne tiefgehende technische Kenntnisse ermöglicht, Roboter und deren Funktionen einfacher zu konfigurieren und zu steuern. Zwei spezielle Programmiersprachen (DSLs) wurden dafür entwickelt: Eine zur Steuerung der Roboterfunktionen und eine andere, um die Roboter mit IoT- und Cloud-Diensten zu verbinden. Die Ergebnisse zeigen, dass dieser Ansatz die Entwicklungszeit und -kosten deutlich reduziert und die Erstellung komplexer robotischer Systeme vereinfacht. Zum Beispiel wurden für ein Testprojekt, bei dem ein Roboter per Fernsteuerung Hindernisse umgehen soll, automatisch Codezeilen generiert. Anstatt manuell Hunderte von Zeilen zu schreiben, generierte das System über 550 Zeilen Code automatisch. Das spart Zeit und vermeidet Fehler, die beim manuellen Programmieren häufig auftreten, wie z.B. Syntaxfehler oder Logikfehler

Im Gegensatz zu dem in der Studie vorgestellten Low-Code-Ansatz, der sich auf die Entwicklung und Konfiguration von robotischen Systemen sowie deren Integration in IoT- und Cloud-Plattformen durch domänenspezifische Sprachen konzentriert, liegt der Schwerpunkt des Konzepts dieser Arbeit auf der Testautomatisierung in IT-Projekten mit Hardwarekomponenten. Dabei ermöglicht eine No-Code-Plattform Fachexpert:innen, ohne Programmierkenntnisse automatisierte Testfälle für Hardware wie Roboterarme zu erstellen und zu verwalten. Während der Low-Code-Ansatz der Studie eher die technische Konfiguration und Vernetzung von Robotern adressiert, betont dieses Konzept eine intuitive Benutzeroberfläche mit Drag-and-Drop-Funktionalitäten und die fortlaufende Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen zur flexiblen Anpassung und Erweiterung der Testfälle.

Die Open Software Platform for Robotic Services (OPRoS) [123] ist eine umfassende Entwicklungsumgebung für Roboteranwendungen, die standardisierte Komponenten und Schnittstellen zur Verfügung stellt. Der No-Code/Low-Code-Ansatz von OPRoS erlaubt es, mithilfe grafischer Editoren und vorgefertigter Module Roboteranwendungen zu erstellen, ohne tiefgehende Programmierkenntnisse zu erfordern. Darüber hinaus bie-

tet die Plattform automatische Testwerkzeuge, die zur Qualitätssicherung von Roboteranwendungen eingesetzt werden. Studien zeigen, dass die Plattform die Entwicklungszeit um bis zu 50 % verkürzen und die Fehlerquote um bis zu 30 % senken kann.

Trotz dieser Effizienzgewinne bestehen jedoch Nachteile in der automatisierten Testgenerierung. Laut Humeniuk et al. [124] können branchenspezifische Randfälle oft nicht ausreichend berücksichtigt werden, da die automatisierten Tests auf vorgegebenen Parametern basieren. Dies erfordert häufig manuelle Anpassungen, um die Robustheit der Tests sicherzustellen und spezielle Anforderungen abzudecken. Im Gegensatz dazu verfolgt diese Arbeit einen Ansatz, der sich nicht ausschließlich auf Roboter fokussiert, sondern auch Schnittstellen zu anderen Systemen wie REST-APIs und mobilen Anwendungen aktiv integriert. Die Plattform ermöglicht Fachexpert:innen über eine No-Code-Oberfläche Testfälle zu erstellen und anzupassen, um projektspezifische Szenarien und Systeminteraktionen abzudecken. Die Kombination automatisierter Prozesse mit dem gezielten Input von Expert:innen erhöht die Flexibilität und Präzision der Testgestaltung und erweitert den Anwendungsbereich über roboterspezifische Anforderungen hinaus.

In der Studie „ROS and Rosbridge: Roboticists Out of the Loop“ [125] besteht die Lösung darin, die Komplexität der Roboterprogrammierung durch die Entkopplung von Roboterentwicklung und experimentellen Schnittstellen zu reduzieren. Das zentrale Element dieser Lösung ist Rosbridge, eine API, die es Entwickler:innen ermöglicht, auf ROS-basierte Roboterfunktionen zuzugreifen, ohne tiefgehendes Wissen über ROS oder Robotik zu benötigen. Rosbridge übersetzt Nachrichten zwischen Robotern und externen Programmen in ein leicht verständliches JSON-Format. Dadurch können Entwickler:innen Webtechnologien wie JavaScript und HTML nutzen, um Roboter zu steuern und mit ihnen zu interagieren. In einem Experiment wurde Rosbridge auch eingesetzt, um großangelegte Mensch-Roboter-Interaktionsstudien über das Internet durchzuführen. Dabei konnten Probanden über eine Webanwendung mit Robotern interagieren und die Ergebnisse der Interaktionen wurden zur Analyse gesammelt. Dies zeigt, wie die Lösung nicht nur die RobotikExpert:innen entlastet, sondern auch eine schnelle und skalierbare Methode zur Entwicklung von Anwendungen und Experimenten bietet. Der wesentliche Unterschied liegt darin, dass Rosbridge eine Schnittstelle bereitstellt, die es ermöglicht, ROS-basierte Roboteranwendungen über Webtechnologien zu entwickeln, ohne tiefere Kenntnisse in der Robotik zu erfordern. Im Gegensatz dazu fokussiert sich der vorgestellte Ansatz auf die Automatisierung von Testfällen für Hardwarekomponenten wie Roboterarme über eine No-Code-Plattform, die Fachexpert:innen ohne Programmierkenntnisse unterstützt. Während Rosbridge eine flexible Lösung für allgemeine Robotikanwendungen bietet, steht hier die spezifische Testautomatisierung im Vordergrund.

Alle besprochenen Studien zeigen signifikante Vorteile bei der Reduzierung der Entwicklungszeit durch den Einsatz von grafischen Benutzeroberflächen, vordefinierten Modulen und der Automatisierung von Aufgaben. In den Arbeiten zu „A Low-Code Approach for Connected Robots“ [122], OPRoS [123] und Rosbridge [125] wird die Entwicklung vereinfacht, sodass weniger technisches Fachwissen benötigt wird. Diese Ansätze fokussieren sich dabei insbesondere auf die Entwicklung und Steuerung von robotischen Systemen sowie deren Integration in IoT-Umgebungen oder Webschnittstellen. Im Vergleich dazu bietet der hier vorgestellte Ansatz eine umfassendere Lösung, die nicht nur auf die Automatisierung von Roboteraufgaben beschränkt ist. Die No-

Code-Plattform ermöglicht die Erstellung und Verwaltung von Testfällen für Hardwarekomponenten, Webanwendungen und mobile Anwendungen. Die Steuerung des Roboterarms ist nur ein kleiner Teil des Gesamtumfangs. Der Fokus liegt auf der flexiblen Anpassung der Testfälle durch Fachexpert:innen, um sowohl Hardware- als auch Softwaretests abzudecken. Dies unterscheidet sich von den besprochenen Studien, die sich hauptsächlich auf Roboteranwendungen konzentrieren, während hier eine breitere Anwendung und eine höhere Flexibilität in der Testgestaltung ermöglicht wird.

5 Motivation und Anforderungen

Die Anforderungsanalyse ist ein kritischer Schritt in der Entwicklung von Softwaresystemen, insbesondere wenn es um die Konzeption innovativer Lösungen geht. Im Fokus steht die Entwicklung eines Konzepts für die Integration von Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess, um komplexe Testfälle mit Hardware- und Softwarekomponenten zu automatisieren. Diese Automatisierung soll durch den Einsatz eines Roboterarms und einer No-Code-Plattform realisiert werden.

In der Praxis sind Entwicklerteams oft so sehr auf die technische Umsetzung von verteilten Software- und Hardwarekomponenten fokussiert, dass die Qualitätssicherung leidet. Zwar könnten Fachexpert:innen manuelle Tests durchführen, doch wäre der Aufwand dabei zu hoch, insbesondere bei Szenarien, die Präzision und Wiederholbarkeit erfordern, wie etwa bei Hardwarekomponenten, wo schon minimale Abweichungen wie ein veränderter Winkel entscheidend sein können.
Die Herausforderung besteht darin, Fachexpert:innen sinnvoll in den Entwicklungsprozess zu integrieren, um ihre Domänenkenntnisse in automatisierte Testprozesse einzubringen. Das hier vorgestellte Konzept bietet einen Ansatz, um diesen Einbezug zu erleichtern und dadurch die Qualität und Effizienz des Testens zu steigern.

Zur Erhebung der notwendigen Anforderungen für ein derartiges Projekt wurden Expert:inneninterviews geführt und tiefgreifende Recherchen angestellt. Die Auswahl der Expert:innen erfolgte gezielt nach ihrer beruflichen Expertise und Erfahrung. Ziel war es, ein breites Spektrum an Wissen und Erkenntnissen zu erfassen, um eine umfassende Anforderungsanalyse zu gewährleisten. Die Auswahlkriterien umfassten eine langjährige Berufserfahrung in relevanten Bereichen sowie spezifische Kenntnisse in der Implementierung und Anwendung von Low-Code/No-Code-Plattformen in Kombination mit automatisierten Systemen. Dies ermöglichte eine differenzierte Betrachtung der relevanten Themenfelder durch ausgewiesene Expert:innen [126]. Diese Gespräche mit Expert:innen, die Erfahrungen mit Model-Driven-Development und verwandten Technologien wie Low-Code und No-Code Plattformen haben, dienten dazu, ein umfassendes Verständnis für das Potenzial dieser Technologien zu entwickeln. Das Ziel besteht darin, Anforderungen zu erheben und diese in Zusammenarbeit mit Fachexpert:innen zu evaluieren sowie gegebenenfalls zu erweitern.

5.1 Methodik der Expert:inneninterviews

In diesem Abschnitt werden die grundlegenden Erkenntnisse aus den Expert:inneninterviews geschildert. Das vollständige Transkript ist in Kapitel 11.3 zu finden. Die Durchführung der Interviews basierte auf einem semi-strukturierten Interviewleitfaden (siehe

Kapitel 11.2), der darauf ausgerichtet war, sowohl qualitative als auch quantitative Fragen zu ermöglichen. Dieser Ansatz sollte sicherstellen, dass einerseits die relevanten Themenbereiche abgedeckt und andererseits tiefergehende Diskussionen gefördert werden, die unerwartete Einsichten und Perspektiven offenlegen können [126]. Die Themenbereiche umfassten unter anderem Erfahrungen mit der Implementierung von MDD-Ansätzen sowie die Bewertung der Effektivität dieser Technologien im Vergleich zu traditionellen Entwicklungsmethoden.

Um die Authentizität und Genauigkeit der gesammelten Daten zu gewährleisten, wurden die Interviews aufgezeichnet und anschließend transkribiert (11.3). Die Auswertung der transkribierten Interviews erfolgte mittels qualitativer Inhaltsanalyse [126] mit dem Ziel, Kernthemen, Muster und wiederkehrende Einsichten zu identifizieren. Diese Methode ermöglichte es, die Vielschichtigkeit der Expert:innenmeinungen zu erfassen und eine fundierte Basis für die Anforderungsanalyse zu schaffen.

Ziel war es, eine strukturierte und zugleich flexible Vorgehensweise zu etablieren, die es ermöglicht, tiefgehende und präzise Informationen zu erheben, die für die Konzeption und Entwicklung einer No-Code-Plattform von entscheidender Bedeutung sind.

Durch diesen methodischen Ansatz wurde eine solide Grundlage für die Anforderungsanalyse geschaffen, die es ermöglicht, die spezifischen Bedürfnisse und Herausforderungen bei der Entwicklung der No-Code-Plattform präzise zu adressieren. Die gewonnenen Erkenntnisse bilden die Basis für die weitere Konzeption und Entwicklung der Plattform mit dem Ziel, eine effiziente, effektive und benutzerfreundliche Lösung zur Automatisierung komplexer Testfälle zu realisieren.

5.2 Erkenntnisse aus den Expert:inneninterviews

Die demografischen Daten der Expert:innen zeigen ein Spektrum an Erfahrungen im Bereich Software-Engineering, Softwarearchitektur und Projektleitung. Die Berufserfahrung der Expert:innen liegt zwischen 20 und 25 Jahren, wobei der Durchschnitt bei 21,3 Jahren liegt. Zwei Expert:innen sind derzeit im E-Health- und Gesundheitssektor tätig, während einer im Bereich der allgemeinen Softwarearchitektur tätig ist. Alle Expert:innen haben langjährige Erfahrung mit vereinfachten Entwicklungsansätzen, einschließlich Domain-Specific-Languages (DSL) und Low-Code-Ansätzen, was ihnen ein tiefes Verständnis sowohl der technischen als auch der organisatorischen Herausforderungen in Softwareprojekten verleiht.

Die Expert:innen betonen den Mehrwert von Model-Driven-Development Ansätzen wie Domain-Specific-Languages und modellbasierten Ansätzen bei der Erleichterung und Beschleunigung des Entwicklungsprozesses. Eine zentrale Erkenntnis ist die wichtige Rolle der Integration von Fachexpert:innen in den Entwicklungsprozess. Diese Integration kann den Entwicklungsprozess erheblich verbessern und die Qualität der Softwarelösungen steigern.

Alle Expert:innen (100 %) waren sich einig, dass die Integration von Fachexpert:innen den Entwicklungsprozess signifikant unterstützt, da sie zu einer präziseren Abbildung der Anforderungen und einer effizienteren Gestaltung des gesamten Ablaufs führt. Ein

Experte erklärte: „Die Einbindung von Fachexpert:innen ist wichtig, um sicherzustellen, dass das System den realen Bedarf abdeckt und um frühzeitig eine Rückmeldung zu erhalten.“ Diese Aussage betont, dass Fachexpert:innen, die über tiefes Domänenwissen verfügen, wertvolle Einblicke geben können, die sicherstellen, dass die entwickelten Lösungen den tatsächlichen Anforderungen entsprechen. Die frühzeitige und kontinuierliche Einbindung von Fachexpert:innen kann Missverständnisse minimieren und die Entwicklung zielgerichteter gestalten.

67 % der Expert:innen betonten die Vorteile einer effizienteren Kommunikationsbasis durch die Integration von Fachexpert:innen. Ein Experte hob hervor, dass die Integration von Fachexpert:innen zu einer effizienteren Kommunikationsbasis führt: „Die Vorteile liegen in der effizienteren Kommunikationsbasis, der einheitlichen gemeinsamen Sprache und der reduzierten Fehleranfälligkeit.“ Diese gemeinsame Sprache zwischen Fachexpert:innen und Entwickler:innen fördert das Verständnis und die Zusammenarbeit, was besonders in interdisziplinären Teams von Vorteil ist. Der Experte betonte zudem: „Diese Aspekte können insbesondere in interdisziplinären Teams, in denen Fachexpert:innen eng mit Entwickler:innen zusammenarbeiten, zu einer deutlichen Effizienzsteigerung führen.“

67 % der Befragten betonten die Bedeutung der Wartung von Testfällen sowie deren kontinuierliche Anpassung an neue Anforderungen. Ein weiterer wichtiger Punkt ist die Verbesserung und Vereinfachung der Wartung und Weiterentwicklung von Testfällen. Ein Experte führte aus: „Die Wartung muss kontinuierlich erfolgen, sonst verlieren die Testfälle schnell ihren Wert. Es ist wichtig, die Fachexpert:innen auch in die Wartung einzubeziehen.“ Dies zeigt, dass Fachexpert:innen nicht nur in der initialen Entwicklungsphase wertvoll sind, sondern auch während des gesamten Lebenszyklus eines Projekts. Ihre kontinuierliche Einbindung stellt sicher, dass die Testfälle relevant und aktuell bleiben.

Alle Expert:innen (100 %) wiesen zudem auf die Notwendigkeit hin, Rahmenbedingungen zu schaffen, die eine effektive Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen ermöglichen. Dies umfasst klare Kommunikationsstrukturen und die Bereitstellung geeigneter Werkzeuge, die sowohl technische als auch fachliche Anforderungen berücksichtigen. Ein Experte betonte, dass direkte Zusammenarbeit selten sei: „Direkte Zusammenarbeit zwischen Entwickler:innen und Fachexpert:innen habe ich kaum erlebt“. Dies zeigt die Notwendigkeit klar definierter Rollen und Werkzeuge, um die Kommunikation zu erleichtern.

Zusammenfassend lassen sich aus den Expert:inneninterviews wichtige Einsichten gewinnen, die für die Konzeption und Entwicklung eines Prototyps einer No-Code-Plattform zur Automatisierung komplexer Testfälle mit Software- und Hardwarekomponenten relevant sind. Die Integration von Fachexpert:innen, die Anwendung vereinfachter Softwareentwicklungsmethoden und die Schaffung effektiver Rahmenbedingungen für die Zusammenarbeit sind Schlüsselelemente, die berücksichtigt werden müssen, um die Ziele des Projekts zu erreichen. Die gewonnenen Erkenntnisse bieten eine solide Grundlage für die weitere Forschung und Entwicklung in diesem Bereich.

5.3 Anforderungen

Die Anforderungen an die Plattform wurden systematisch durch Expert:inneninterviews sowie eine umfassende Literatur- und Marktrecherche ermittelt. Diese zweigleisige Analyse gewährleistet, dass die Plattform sowohl die spezifischen Bedürfnisse der Zielgruppe als auch aktuelle Standards und bewährte Methoden der Testautomatisierung adressiert. Ziel ist es, eine Lösung zu entwickeln, die für Fachexpert:innen ohne technische Programmierkenntnisse nutzbar ist und sich nahtlos in bestehende Entwicklungsprozesse einfügt.

Ein zentrales Ergebnis der Anforderungsanalyse war die Entscheidung für eine No-Code-Plattform. Die Wahl dieses Ansatzes statt beispielsweise BDD (Behavior-Driven-Development), KDT (Keyword-Driven-Testing), Low-Code oder einer Domain-Specific-Language (DSL), basiert auf den spezifischen Nutzeranforderungen und den Erfahrungen der Interviewpartner. No-Code ermöglicht Fachexpert:innen ohne Programmierkenntnisse, Testfälle unkompliziert zu erstellen und anzupassen, während Low-Code- und DSL-Ansätze fortgeschrittene technische Kenntnisse erfordern und somit nicht ideal für die Zielgruppe sind [127]. Die Befragten betonten hierbei die Relevanz einer intuitiven und klar strukturierten Oberfläche, da „die Integration von Fachexpert:innen wichtig ist, um sicherzustellen, dass das System den realen Bedarf abdeckt.“ Eine einfache Bedienbarkeit ohne technische Hürden ist ein entscheidender Vorteil der No-Code-Lösung, um eine breite Akzeptanz und einfache Handhabung sicherzustellen [128].

Die in den Interviews identifizierten funktionalen Anforderungen (Tabelle 2) zielen auf eine breite Anwendbarkeit der Plattform ab. Dazu gehört die Testfallverwaltung (FA1), die es Anwender:innen ermöglicht, Testfälle zu erstellen, zu bearbeiten, zu speichern und zu löschen. Anwender:innen können neue Testfälle anlegen und Testschritte aus einer Elementbibliothek per Drag-and-Drop hinzufügen. Die Anpassung von Testschritten (FA2) erlaubt es Nutzern, innerhalb eines Testfalls die Reihenfolge von Testschritten zu ändern, Testschritte hinzuzufügen oder zu entfernen und spezifische Parameter für jeden Testschritt einzugeben. Die Speicherfunktionen (FA3) gewährleisten, dass Änderungen sowohl manuell als auch automatisch gespeichert werden können, um Datenverlust zu vermeiden.

Die Erweiterung der Testschritte (FA4) ermöglicht Anwender:innen, neue Testschritte für den Testeditor zu erstellen, indem sie Titel, Beschreibung und Kategorie definieren. Für jedes neu erstellte Element wird ein Ticket in einem Ticket Management System erstellt. Gleichzeitig soll der angelegte Testschritt sofort für die Anwender:innen nutzbar sein, auch bevor die technische Umsetzung abgeschlossen ist, was die Flexibilität und Anpassbarkeit der Tests erhöht. Die Informationsanzeige bei Testschritten (FA5) unterstützt die selbstständige Bedienbarkeit, indem Kurzbeschreibungen beim Mouse-Over und detaillierte Informationen sowie Anwendungsbeispiele im integrierten Wiki (FA10) verfügbar sind. Testschritte können Kategorien zugeordnet werden, um die Bibliothek zu strukturieren.

Der Export von Testfällen (FA6) ermöglicht es, erstellte Testfälle in ein ausführbares Format zu exportieren, das mit dem Robot Framework kompatibel ist. Dies stellt sicher, dass die Testfälle außerhalb der Plattform erfolgreich ausgeführt werden können. Die

Ausführung von Testfällen (FA7) direkt aus der Plattform heraus, indem eine vorbereitete CI-Pipeline (Continuous Integration) gestartet wird, macht den Testprozess effizienter und gibt den Anwender:innen unmittelbares Feedback zu ihren erstellten Tests (siehe Kapitel 2.10). Ein zentrales Dashboard für Testausführungen (FA8) bietet einen Überblick über aktuelle und vergangene Testläufe mit detaillierten Statusinformationen wie „läuft“, „fehlgeschlagen“ oder „erfolgreich“. Das Dashboard liefert detaillierte Informationen zu jedem Testlauf einschließlich Datum, Dauer und beteiligten Testfällen, was die Überwachung und Analyse der Tests erleichtert.

Die Fehleranalyse und Bugtracking (FA9) verbessern den Entwicklungsprozess, indem ein automatischer Fehleranalysator Fehler während der Testausführung kategorisiert und dem Anwender:innen Empfehlungen gibt, ob er selbst Änderungen vornehmen kann oder Entwickler:innen Unterstützung benötigt. Zudem ermöglicht die Plattform die direkte Erstellung von Bugtickets in externen Bugtracking-Systemen, wobei relevante Fehlerinformationen automatisch übernommen werden. Die Wiki-Integration (FA10) bietet eine zentrale Wissensdatenbank für Anleitungen und Beschreibungen der Testschritte. Anwender:innen können Wikiseiten erstellen, bearbeiten und kommentieren, wobei eine Versionierung der Wikiseiten gewährleistet ist, um Änderungen nachvollziehen zu können. Die Bereitstellung vereinfachter Logs (FA11) stellt sicher, dass Anwender:innen verständliche Logfiles mit detaillierten Informationen zu den ausgeführten Schritten eines Testlaufs erhalten. Die Logfiles werden in einem für Anwender:innen verständlichen Format präsentiert, beispielsweise durch Klartextbeschreibungen statt technischer Fehlermeldungen, was die Analyse und Fehlerbehebung erleichtert.

Die nicht-funktionalen Anforderungen (Tabelle 3) ergänzen die funktionalen Anforderungen und legen Qualitätskriterien fest, die für den langfristigen, effizienten Einsatz der Plattform wesentlich sind. An erster Stelle steht die Bedienbarkeit (NFA1), um sicherzustellen, dass auch Fachexpert:innen ohne technische Vorkenntnisse die Plattform problemlos nutzen können. Eine webbasierte Benutzeroberfläche (NFA2) gewährleistet den einfachen Zugriff auf alle Funktionen ohne zusätzliche Softwareinstallation, wodurch die Zugänglichkeit und Benutzerfreundlichkeit erhöht werden. Außerdem gewährleistet die Kompatibilität mit aktuellen Webbrowsern und ein responsives Design gewährleisten eine optimale Darstellung auf verschiedenen Geräten (Desktop, Tablet, Smartphone) und eine nahtlose Integration in bestehende Systeme und Tools. Die Erweiterbarkeit (NFA3) stellt sicher, dass neue Funktionen und Technologien ohne grundlegende Änderungen am Kernsystem integriert werden können. Die Plattform darf nicht auf ein spezifisches Framework beschränkt sein, um eine breitere Flexibilität und Integration in unterschiedliche Entwicklungsumgebungen zu gewährleisten. Das System muss verschiedene domänenspezifische Sprachen (DSLs) unterstützen, auch solche, die speziell für einen Anwendungsfall entwickelt worden sind. Diese Sprachen kommen ausschließlich im Hintergrund zum Einsatz, ohne dass der Benutzer sie selbst schreiben oder verwalten muss. Die Implementierung und Nutzung der DSLs erfolgt dabei im Hintergrund, sodass der Benutzer sich auf die Anwendung konzentrieren kann.

Eine robuste Fehlerbehandlung (NFA4) ist entscheidend, um Anwender:innen bei Problemen zu unterstützen und Entwickler:innen über notwendige Maßnahmen zu informie-

ren. Eine umfassende Dokumentation (NFA5) für Anwender:innen und Entwickler:innen erleichtert den Einstieg und die fortlaufende Nutzung der Plattform, unterstützt durch regelmäßig aktualisierte Schulungsmaterialien.

Zusammengefasst bildet die Verbindung aus Anforderungen aus den Expert:inneninterviews und bewährten Methoden aus der Literatur die Grundlage für die Wahl einer No-Code-Plattform. Dieser Ansatz ermöglicht eine unkomplizierte, skalierbare und zukunftssichere Testautomatisierungslösung, die den Arbeitsaufwand für technische Anpassungen minimiert und Fachexpert:innen eine direkte Beteiligung am Testprozess eröffnet.

Nr.	Anforderung
FA1	**Testfallverwaltung**: Die Plattform muss es Anwender:innen ermöglichen, Testfälle zu erstellen, zu bearbeiten, zu speichern und zu löschen. Anwender:innen sollen neue Testfälle anlegen und Testschritte aus einer Elementbibliothek per Drag-and-Drop hinzufügen können.
FA2	**Anpassung von Testschritten:** Die Plattform muss es Anwender:innen ermöglichen, innerhalb eines Testfalls die Reihenfolge von Testschritten zu ändern, Testschritte hinzuzufügen oder zu entfernen und spezifische Parameter für jeden Testschritt einzugeben.
FA3	**Speicherfunktionen**: Die Plattform muss sowohl manuelles Speichern über einen Speichern-Button als auch automatisches Speichern bei jeder Änderung eines Testfalls oder Testschritts ermöglichen, um Datenverlust zu vermeiden.
FA4	**Erweiterung der Testschritte**: Die Plattform soll Anwender:innen ermöglichen, neue Testschritte zu erstellen, indem sie Titel, Beschreibung und Kategorie definieren. Für jedes neu erstellte Element generiert die Plattform automatisch ein Ticket in einem Ticket Management System. Gleichzeitig soll das Element sofort für die Anwender:innen nutzbar sein, auch bevor die technische Umsetzung abgeschlossen ist. Testschritte sollen Kategorien zugeordnet werden können, um die Bibliothek zu strukturieren.
FA5	**Informationsanzeige bei Testschritten**: Die Plattform muss sicherstellen, dass jeder Testschritt in der Bibliothek beim Mouse-Over eine Kurzbeschreibung anzeigt.
FA6	**Export von Testfällen**: Die Plattform muss die Möglichkeit bieten, erstellte Testfälle in ein ausführbares Format zu exportieren, das mit dem Robot Framework kompatibel ist. Der Export muss alle notwendigen Dateien und Konfigurationen enthalten, um den Testfall außerhalb der Plattform erfolgreich ausführen zu können.
FA7	**Ausführung von Testfällen:** Die Plattform muss es Anwender:innen ermöglichen, Testfälle direkt aus der Plattform heraus auszuführen, indem eine vorbereitete CI-Pipeline gestartet wird. Anwender:innen sollen den Ausführungsprozess initiieren können, ohne die Plattform zu verlassen.
FA8	**Dashboard für Testausführungen**: Die Plattform muss ein zentrales Dashboard bereitstellen, das aktuelle und vergangene Testausführungen mit Statusinformationen wie „läuft“, „fehlgeschlagen“ oder „erfolgreich“ anzeigt. Das Dashboard muss detaillierte Informationen zu jedem Testlauf bieten, einschließlich Datum, Dauer und beteiligten Testfällen.

FA9	**Fehleranalyse und Bugtracking**: Die Plattform muss einen automatischen Fehleranalysator enthalten, der Fehler während der Testausführung kategorisiert und dem Anwender:innen Empfehlungen gibt, ob er selbst Änderungen vornehmen kann oder Entwickler:innen unterstützung benötigt. Anwender:innen müssen aus dem Dashboard heraus direkt ein Bugticket in einem externen Bugtracking-System erstellen können, wobei relevante Fehlerinformationen automatisch übernommen werden.
FA10	**Wiki-Integration**: Die Plattform muss ein integriertes Wiki bereitstellen, das Anleitungen und Beschreibungen der Testschritte enthält. Anwender:innen sollen Wikiseiten erstellen, bearbeiten und kommentieren können. Eine Versionierung der Wikiseiten muss gewährleistet sein, um Änderungen nachvollziehen zu können.
FA11	**Vereinfachte Logs**: Die Plattform muss Logfiles bereitstellen, die detaillierte Informationen zu den ausgeführten Schritten eines Testlaufs enthalten. Die Logfiles sollen in einem für Anwender:innen verständlichen Format präsentiert werden, beispielsweise durch Klartextbeschreibungen statt technischer Fehlermeldungen.
FA12	**Benachrichtigungssystem**: Die Plattform muss ein effektives Benachrichtigungssystem bereitstellen. Entwickler:innen sollen benachrichtigt werden, wenn relevante Ereignisse wie Wiki-Kommentare oder kritische Fehler auftreten. Anwender:innen sollen automatisch informiert werden, wenn Tests fehlgeschlagen sind oder neue Testschritte benötigt werden, um zeitnah reagieren zu können.

Tabelle 2 Funktionale Anforderungen an die Plattform

Nr.	Anforderung
NFA1	**Bedienbarkeit**: Die Plattform muss eine intuitive Benutzeroberfläche bereitstellen, die eine einfache Bedienung ermöglicht. Durch eine Drag-and-Drop-Funktionalität, ein übersichtliches Dashboard und leicht zugängliche Speicheroptionen sollen Anwender:innen ohne umfangreiche Einarbeitung effektiv arbeiten können.
NFA2	**Webbasierte Benutzeroberfläche:** Die Plattform muss mit aktuellen Versionen gängiger Webbrowser (z. B. Chrome, Firefox, Safari, Edge) kompatibel sein und über eine webbasierte Benutzeroberfläche verfügen, die ohne zusätzliche Softwareinstallation zugänglich ist. Sie muss ein responsives Design bieten, das auf verschiedenen Geräten (Desktop, Tablet, Smartphone) eine optimale Benutzererfahrung ermöglicht.
NFA3	**Erweiterbarkeit**: Die Plattform muss modular aufgebaut sein, um zukünftige Erweiterungen und Integrationen neuer Funktionen und Technologien zu ermöglichen, ohne grundlegende Änderungen am Kernsystem vorzunehmen. Sie darf nicht auf ein spezifisches Framework wie Robot Framework beschränkt sein. Es muss möglich sein, verschiedene Domain-spezifische Sprachen (DSLs), einschließlich benutzerdefinierter DSLs, zu verwenden.
NFA4	**Fehlerbehandlung**: Die Plattform muss eine robuste Fehlerbehandlung bieten, indem sie klare und verständliche Fehlermeldungen anzeigt und Anwender:innen bei der Fehlerbehebung unterstützt. Sie soll gegenüber unerwarteten Eingaben widerstandsfähig sein und bei Systemfehlern ein definiertes Verhalten zeigen.

NFA5	**Dokumentation**: Es muss eine umfassende und aktuelle Dokumentation für Anwender:innen und Entwickler:innen bereitgestellt werden, die alle Funktionen und Prozesse der Plattform beschreibt. Schulungsmaterialien wie Tutorials oder Anleitungen sollen den Einstieg erleichtern und regelmäßig aktualisiert werden, um Neuerungen und Updates abzudecken.

Tabelle 3 Nicht funktionale Anforderungen an die Plattform

6 Entwurf und Implementierung

Das vorgestellte Konzept basiert auf einer Low-Code-Technologie und überführt diese Grundlage in eine No-Code-Plattform. Diese Plattform wird nahtlos in bestehende Systeme wie Bugtracking-Tools (z. B. Jira[20]), CI/CD-Pipelines und ein Wiki (z. B. Confluence[21]) integriert, um Fachexpert:innen ohne Programmierkenntnisse in die Qualitätssicherung einzubinden. Ziel ist, dass Fachpersonen ihr Wissen durch eigenständig erstellte und automatisierte Testfälle einbringen, während Entwickler:innen über eine erweiterbare Befehlsbasis neue Testmöglichkeiten bereitstellen.

Abschnitt 6.1 erläutert die Plattform aus der Sicht der Fachexpert:innen und zeigt, wie sie automatisierte Tests erstellen, ausführen und analysieren können. In Abschnitt 6.2 wird die zugrunde liegende Systemarchitektur beschrieben, einschließlich der Anbindung externer Systeme. Abschnitt 6.3 illustriert anhand eines Prototyps, wie die theoretisch entwickelten Konzepte praktisch umgesetzt werden. Abschnitt 6.4 beschäftigt sich mit der Integration in CI/CD-Pipelines, um Testabläufe kontinuierlich und automatisiert in den Entwicklungsprozess einzubinden. In Abschnitt 6.6 wird beschrieben, wie Fehler mithilfe automatisierter Log-Analyse frühzeitig erkannt, klassifiziert und anschließend in verständlicher Form für Fachexpert:innen aufbereitet werden.
Abschließend erläutert Kapitel 6.7, wie die Plattform erweiterbar bleibt, indem Entwickler:innen neue Befehle hinzufügen und so auf veränderte Anforderungen reagieren können.

6.1 Nutzung der Plattform aus Sicht der Fachexpert:innen

Die No-Code-Plattform wurde entwickelt, um Fachexpert:innen ohne Programmierkenntnisse die Möglichkeit zu geben, automatisierte Testfälle für IT-Projekte zu erstellen, auszuführen und zu analysieren. Die Plattform implementiert dabei verschiedene Ansätze der Integration und unterstützt z.B. Webanwendungen, Mobile Apps oder externe Hardwarekomponenten.

Die Hauptansicht der Plattform ist der Testfall-Editor. Er verfügt über eine Drag-and-Drop-Oberfläche (siehe Abbildung 8) und links daneben befindet sich die Befehlspalette, die in verschiedene Kategorien wie „App Interaktion“ unterteilt ist. Jeder Befehl repräsentiert einen vordefinierten Testschritt, zum Beispiel „Starte App auf iOS-Smartphone“.

[20] https://www.atlassian.com/de/software/jira
[21] https://www.atlassian.com/de/software/confluence

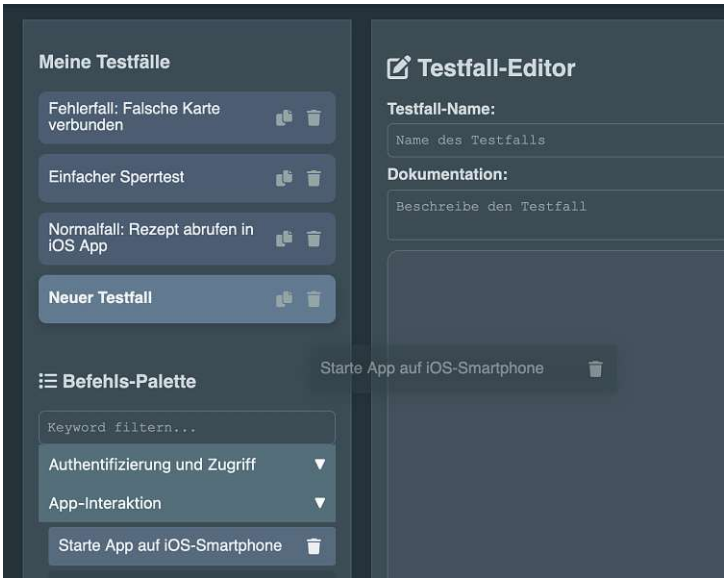

Abbildung 8 Testfalleditor mit Drag & Drop Funktionalität

Die Fachperson kann die gewünschten Testschritte per Drag-and-Drop in den zentralen Arbeitsbereich ziehen und so die Sequenz der Testschritte festlegen. Es gibt unparametrisierte und parametrisierte Befehle. Unparametrisierte Befehle können direkt per Drag-and-Drop eingefügt und verwendet werden. Bei parametrierten Befehlen müssen spezifische Parameter eingegeben werden, wie zum Beispiel die spezifische Versionsnummer der App, die gestartet werden soll. Dies ermöglicht eine präzise Definition des Testfalls entsprechend den Anforderungen.

Bestehende Testfälle können jederzeit angepasst werden. Die Reihenfolge der Testschritte lässt sich einfach durch Verschieben der Befehle mit den Richtungspfeilen oder per Drag & Drop ändern. Testschritte können hinzugefügt oder entfernt werden, um den Testfall an neue Bedingungen anzupassen (siehe Abbildung 9). Sollte ein benötigter Befehl nicht in der Bibliothek vorhanden sein, kann die Fachperson über die Funktion einen neuen Befehl anfordern (siehe Abbildung 10). Dabei wird Titel, Beschreibung und Kategorie des Befehls angegeben. Das System generiert automatisch ein Ticket für das Entwicklerteam, welches dann die notwendige Logik hinter dem Befehl implementiert. In der Zwischenzeit kann der Befehl bereits im Testfall verwendet werden, um den Arbeitsfluss nicht zu unterbrechen.

Abbildung 9 Beispiel für einen zusammengestellten Testfall und den Optionen für die Testschritte

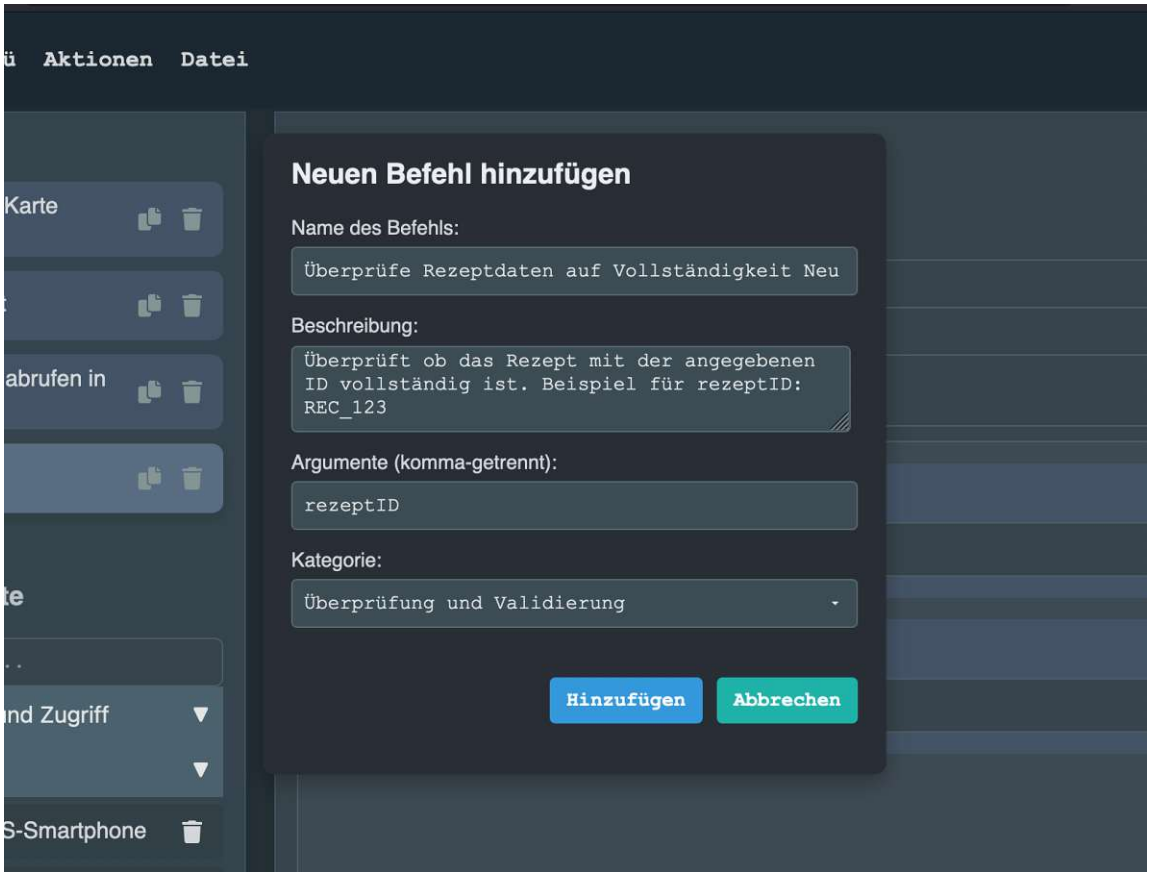

Abbildung 10 Dialogansicht „Neuen Befehl hinzufügen“

Nach der Erstellung kann die Fachperson den Testfall direkt in der Plattform ausführen. Mit einem Klick auf „Testfall ausführen“ wird eine CI -Pipeline gestartet, die den Testfall automatisiert auf einer vorbereiteten Teststage durchführt. Der Fortschritt wird in Echtzeit im Dashboard angezeigt (siehe Abbildung 11).

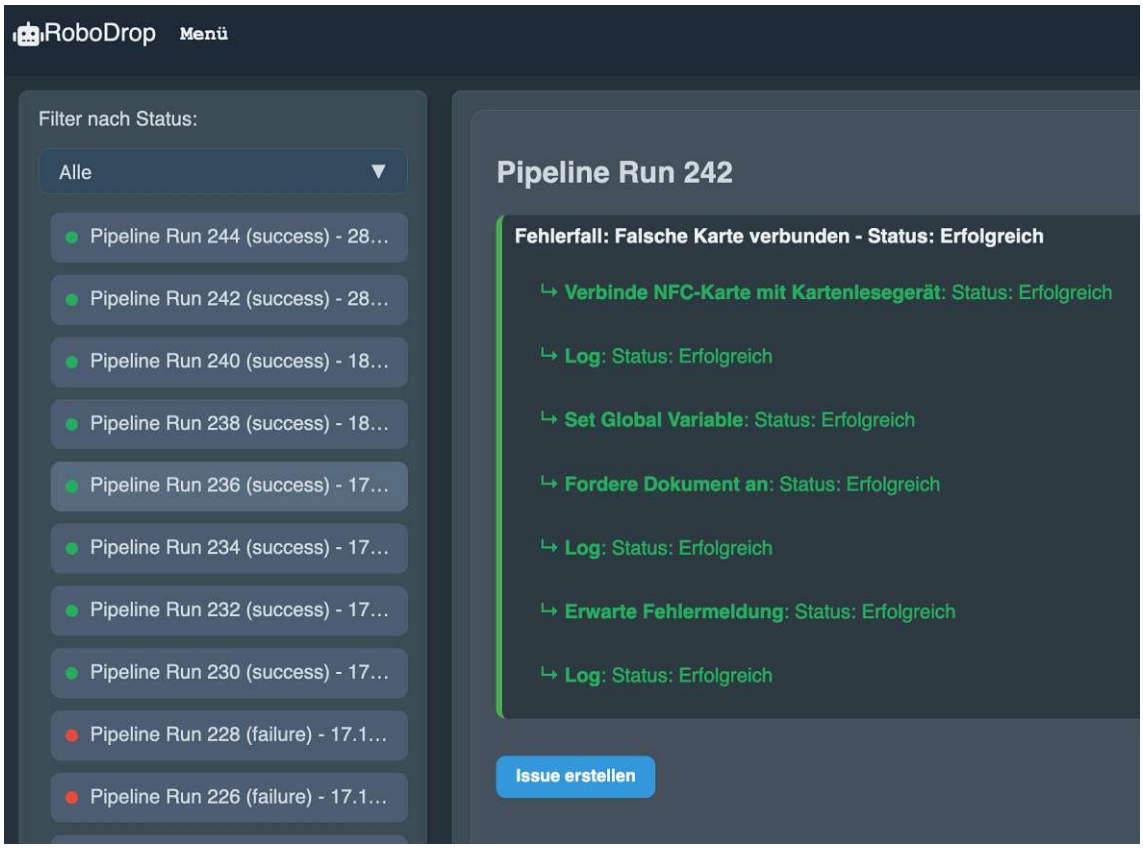

Abbildung 11 Dashboard für ausgeführte Testfälle auf der CI-Pipeline

Das Dashboard zeigt den Fortschritt und Status der Tests, vereinfacht die Fehleranalyse und ermöglicht das Erstellen von Fehlermeldungen (Tickets/Issues) direkt aus der Plattform (siehe Abbildung 12). Vereinfachte Logs bieten detaillierte Informationen in verständlicher Sprache ohne technische Fehlermeldungen. Fachexpert:innen sollen anhand der Beschreibung erkennen können, ob selbst Anpassungen vorgenommen werden können oder ob Unterstützung benötigt wird. Über das Dashboard kann ein Bugticket in einem externen Bugtracking-System erstellt werden, wobei relevante Fehlerinformationen automatisch übernommen werden, wie die Verlinkung zu der konkreten Testausführung.

Abbildung 12 Beispiel einer fehlgeschlagenen Testfallausführung und der Möglichkeit, direkt in der Plattform ein Ticket zu erstellen

Die Plattform fördert die Zusammenarbeit durch ein integriertes Wiki, in dem Anleitungen, Beschreibungen der Befehle und Best Practices bereitgestellt werden (siehe Abbildung 13). Fachexpert:innen können Wikiseiten erstellen, bearbeiten und kommentieren, um ihr Wissen zu teilen und gemeinsam Lösungen zu erarbeiten. Eine Versionierung der Wikiseiten stellt sicher, dass Änderungen nachvollziehbar bleiben.

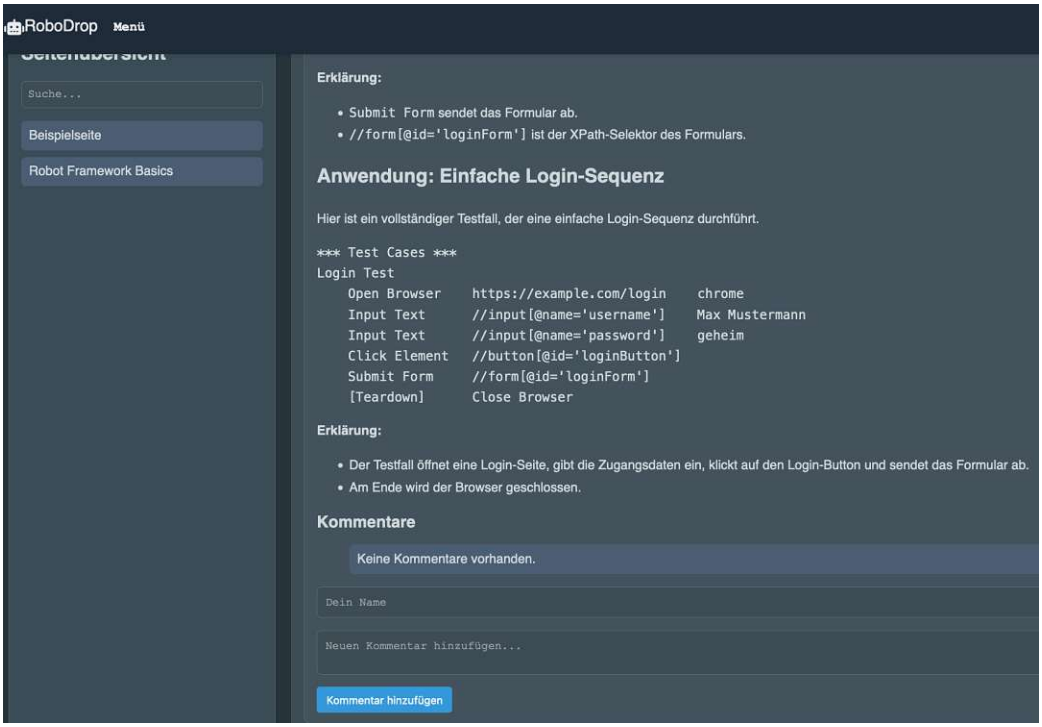

Abbildung 13 Integriertes Wiki als Knowledgebase für den Austausch untereinander und mit dem Entwicklerteam

Ein Beispiel für die Nutzung der Plattform könnte wie folgt aussehen: Eine Fachperson möchte prüfen, ob verschiedene Kartentypen korrekt vom Kartenleser erkannt werden. Die Befehle „Verbinde Karte mit Kartenleser“ und „Prüfe Karte erkannt“ werden in den Testfall-Editor gezogen und die entsprechenden Kartentypen als Parameter werden eingegeben. Nachdem der Testfall ausgeführt wurde, werden die Ergebnisse im Dashboard analysiert und es stellt sich heraus, dass bei einem Kartentyp ein Tippfehler bei der Parametereingabe aufgetreten ist. Durch die vereinfachten Logs wird erkannt, dass eine Anpassung des Testschritts erforderlich ist und der Testfall wird entsprechend angepasst und noch einmal ausgeführt. Nun läuft der Testfall erfolgreich durch.

Durch diese anwenderzentrierte Gestaltung ermöglicht die Plattform Fachexpert:innen eine effiziente und selbstständige Arbeit bei der Testautomatisierung, ohne dass technische Hürden den Prozess erschweren. Die intuitive Benutzeroberfläche, die direkte Ausführung von Testfällen und die umfangreichen Kollaborationsfunktionen tragen wesentlich zur Optimierung der Entwicklungsprozesse bei.

6.2 Systemarchitektur

Das System besteht aus einem Frontend, einem Backend und einer Datenbank. Das Frontend stellt eine Weboberfläche bereit, die über einen beliebigen Webbrowser aufgerufen wird. Fachexpert:innen können dort logische Abläufe oder Testfälle per Drag & Drop definieren sowie bestehende Abläufe anpassen. Über dasselbe Frontend werden zudem Ausführungsergebnisse eingesehen und Tickets für Bugs oder Funktionswünsche in einem extern angebundenen Bug-Tracking-System (etwa Jira[22]) erstellt.

Das Backend setzt sich aus einem Webserver und einem Applikationsserver zusammen. Es übernimmt die Steuerungslogik, nimmt Anfragen vom Frontend entgegen und verarbeitet sie. Eine relationale Datenbank verwaltet die relevanten Daten, darunter definierte Abläufe und Referenzen zu vorhandenen Tickets. Der Zugriff auf die Datenbank erfolgt in der Regel über JDBC/SQL. Über das Backend ist zudem ein Wiki integriert, das als Wissensbasis dient. Hier können Fachexpert:innen Dokumentationen, Prozessbeschreibungen oder Best Practices ablegen. Abbildung 14 zeigt, dass das Wiki direkt mit dem Server kommuniziert, beispielsweise um neu erstellte Inhalte zu speichern oder beim Abruf von Wiki-Seiten die entsprechenden Daten bereitzustellen.

Für die Interaktion mit externen Systemen kommen überwiegend standardisierte Protokolle wie HTTPS und REST zum Einsatz. Über diese Schnittstellen wird beispielsweise ein CI/CD-Orchestrator (z. B. Jenkins[23] oder GitLab CI[24]) angebunden. Sobald ein neuer Commit vorliegt, ist es möglich, vordefinierte Abläufe oder Tests automatisiert auszuführen, sofern die dazugehörigen Skripte oder No-Code-Anweisungen hinterlegt sind. Diese Skripte liegen in der Regel in einem File Repository, das außerhalb der Plattform betrieben wird. Nach Abschluss eines automatisierten Durchlaufs können die

[22] https://www.atlassian.com/de/software/jira
[23] https://www.jenkins.io
[24] https://docs.gitlab.com/ee/ci/

Ergebnisse über REST- oder HTTPS-Aufrufe an die Plattform übermittelt werden, sodass Fachexpert:innen sie über das Frontend abrufen können.

Das Bug-Tracking-System ist ein weiteres externes Tool, in dem Fehlerberichte und Änderungen verwaltet werden. Die Plattform stellt eine Schnittstelle bereit, über die bei Bedarf Tickets erstellt werden können. Die Fachexpert:innen müssen dazu nicht das Bug-Tracking-System selbst aufrufen, sondern können den Vorgang direkt in der No-Code-Oberfläche initiieren. Die so erzeugten Einträge werden anschließend im externen System weitergeführt.

Abbildung 14 verdeutlicht, dass Frontend, Backend, Datenbank und Wiki gemeinsam den Kern der Plattform bilden, während CI/CD-Orchestrator, File Repository und Bug-Tracking-System separat angeordnet sind. Zwischen der Plattform und den externen Komponenten bestehen standardisierte Kommunikationskanäle auf Basis von HTTPS und REST. Innerhalb der Plattform wird eine verschlüsselte Verbindung (HTTPS) zwischen Weboberfläche und Webserver verwendet, während der Applikationsserver per JDBC/SQL auf die Datenbank zugreift. Das Wiki ist als Teil des Backends implementiert und wird über eine interne Schnittstelle angesprochen. Diese Aufteilung gewährleistet eine klare Trennung zwischen Plattformfunktionalitäten und externen Diensten, was sowohl für die Wartung als auch für spätere Erweiterungen von Bedeutung sein kann.

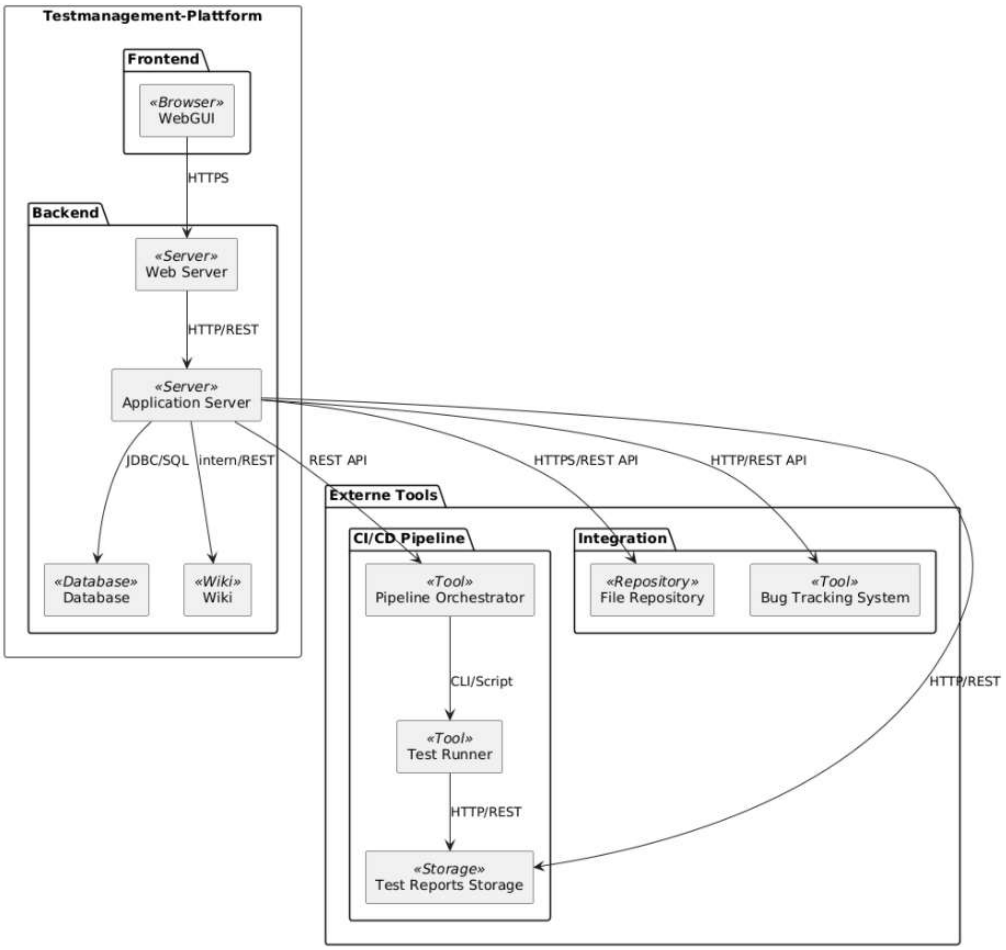

Abbildung 14 Komponentendiagramm der Systemarchitektur

6.3 Prototyp der No-Code Plattform

Im Rahmen dieser Arbeit wurde ein Prototyp einer No-Code-Plattform entwickelt, der es Fachexpert:innen ermöglicht, automatisierte Testfälle für komplexe Projekte mit Software und Hardwarekomponenten zu erstellen, zu verwalten und auszuführen, ohne dass Programmierkenntnisse erforderlich sind. Die Benutzeroberfläche ist so gestaltet, dass sie durch eine Drag-and-Drop-Funktionalität zu bedienen ist. Der zentrale Mechanismus zur Erstellung von Testfällen ist die visuelle Anordnung von vordefinierten Befehlen, die unter anderem bestimmte Aktionen des Roboterarms repräsentieren. Diese können von den Nutzern per Drag-and-Drop zu einem vollständigen Testfall zusammengestellt werden. Die Befehle werden vom Entwicklerteam bereitgestellt und durch Kategorisierung und Filterung für eine einfache Auswahl optimiert.

Die prototypische Webanwendung wurde mit JavaScript, HTML und CSS entwickelt und ist einfach gehalten. Über eine grafische Benutzeroberfläche können Nutzer Testfälle erstellen und verwalten. Die Daten werden im lokalen Speicher des Browsers (LocalStorage) gespeichert, was eine schnelle und unkomplizierte Speicherung ohne zusätzlichen Serveraufwand ermöglicht. Die Basis bildet das Robot Framework (siehe Kapitel 3.4), da es eine etablierte, erweiterbare und vielseitige Plattform für die Testautomatisierung bietet. Ein zentraler Entscheidungsfaktor war die umfangreiche Bibliotheksunterstützung, die es ermöglicht, eine Vielzahl von Anforderungen abzudecken, darunter API-Calls, Selenium für Webtests und Appium für mobile Anwendungen. Diese eingebaute Funktionalität reduziert den Entwicklungsaufwand erheblich und ermöglicht eine schnelle Implementierung. Zudem zeichnet sich das Robot Framework durch seine klare Syntax aus, die eine einfache Anwendung und Anpassung insbesondere für das Entwicklerteam ermöglicht. Diese Eigenschaften machen es ideal, um als Grundlage für eine No-Code-Plattform zu dienen, die sowohl Flexibilität als auch Bedienbarkeit gewährleistet.

Abbildung 15 zeigt die Benutzeroberfläche des Prototyps. Im linken oberen Bereich befindet sich eine Liste mit den vorhandenen Testfällen, die der Benutzer:innendurch Anklicken auswählen oder löschen kann. Darunter befindet sich die Befehlspalette, welche in verschiedene Kategorien unterteilt ist, wie beispielsweise „App Interaktion“. Durch Anklicken einer Kategorie werden die dazugehörigen Befehle angezeigt. Diese Befehle können per Drag-and-Drop in den Testfall-Editor auf der rechten Seite gezogen werden, um Testfälle zu erstellen oder zu bearbeiten.

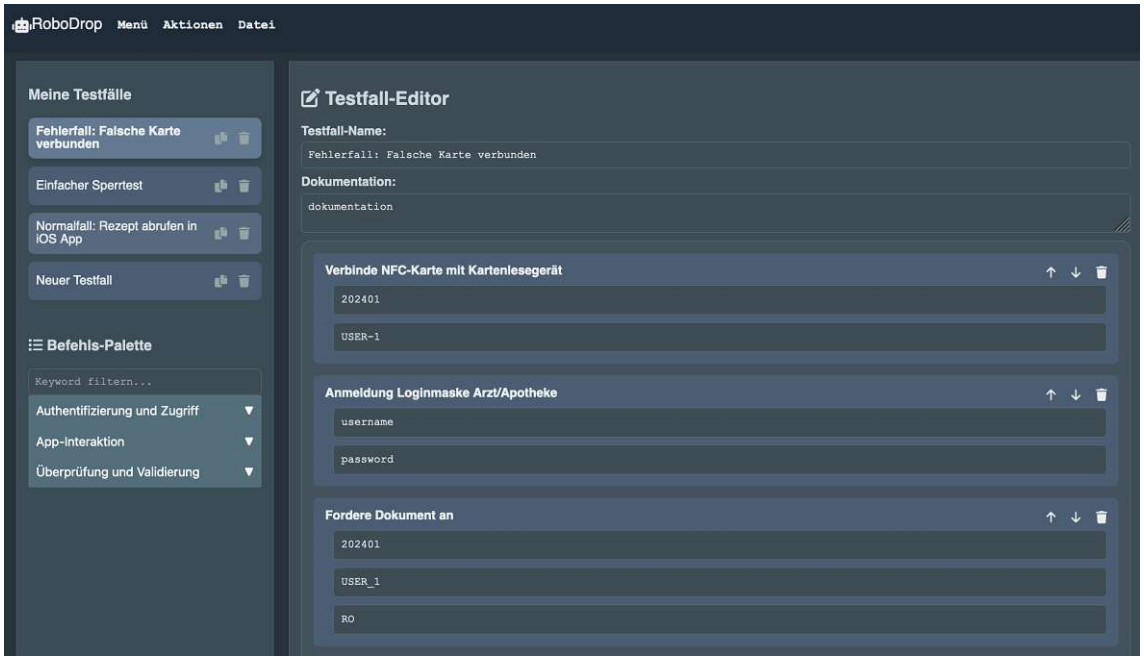

Abbildung 15 Testfalleditor Ansicht des Prototyps

Im rechten Bereich der Benutzeroberfläche ist der Testfall-Editor dargestellt. Hier können Testfälle benannt und dokumentiert werden. Darunter werden die bereits hinzugefügten Testschritte als Blöcke dargestellt. Jeder Testschritt enthält eine Beschreibung sowie Verwaltungsmöglichkeiten, um die Reihenfolge der Schritte zu ändern oder einen Schritt zu löschen.

Oben im Menü unter „Aktionen“ können neue Testfälle oder neue Befehle hinzugefügt werden, was dem Benutzer:innen ermöglicht, die Testumgebung dynamisch anzupassen.

Um den Drag-and-Drop-Mechanismus zu ermöglichen, wird eine HTML5-basierte Lösung implementiert. Abbildung 16 zeigt beispielhaften Code, der die Drag-and-Drop-Funktion für die Testfallerstellung realisiert. Benutzer:innen können Testschritte aus einer Befehlspalette ziehen und in den Testfallbereich fallen lassen:

```
<div id="commandPalette">
  <div class="command" draggable="true" ondragstart="drag(event)"
id="moveCommand">Bewegen</div>
  <div class="command" draggable="true" ondragstart="drag(event)"
id="stopCommand">Anhalten</div>
</div>

<div id="testCaseStructure" ondrop="drop(event)" ondragover="al-
lowDrop(event)">
  <p>Füge hier Befehle ein, um den Testfall zu erstellen...</p>
</div>

<script>
function allowDrop(ev) {
  ev.preventDefault();
}

function drag(ev) {
```

```
  ev.dataTransfer.setData("text", ev.target.id);
}

function drop(ev) {
  ev.preventDefault();
  var data = ev.dataTransfer.getData("text");
  var nodeCopy = document.getElementById(data).cloneNode(true);
  ev.target.appendChild(nodeCopy);
}
</script>
```

Abbildung 16 Beispielimplementierung HTML5 Drag & Drop Funktionalität

Durch diesen Mechanismus können Benutzer:innen komplexe Testfälle erstellen, indem sie visuell einzelne Befehle aus der Befehlspalette auswählen und in den Testfallbereich einfügen. Diese Befehle, die für den Roboterarm relevante Aktionen wie Bewegen, Anhalten oder Greifen repräsentieren, werden vom System als Befehle verwaltet. Diese Testschritte sind in Kategorien unterteilt und können nach verschiedenen Kriterien gefiltert werden. Eine mögliche Datenstruktur für einen Befehl könnte wie folgt aussehen (Abbildung 17). Diese Struktur ist nicht zwingend an eine Datenbank gebunden, sondern könnte auch auf einer gemeinsamen Basis, wie beispielsweise einer Datei in einer bestimmten Version realisiert werden.

```
{
  "command": "Verbinde Karte mit Kartenleser",
  "description": "Beispiel: Verbinde Karte „Ordinationskarte“ mit dem
Kartenleser",
  "category": "Bewegung",
  "arguments": ["karte"]
}
```

Abbildung 17 Beispiel wie ein Befehl in einer Datenbank hinterlegt werden könnte

Das Laden von Befehlen erfolgt in der prototypischen Plattform beispielsweise durch den folgenden Mechanismus. Der folgende JavaScript-Code zeigt, wie neue Befehle importiert und in die entsprechende Kategorie einsortiert werden (Abbildung 18):

```
<div id="commandPalette"></div>

<div id="testCaseStructure" ondrop="drop(event)" ondragover="al-
lowDrop(event)">
  <p>Füge hier Befehle ein...</p>
</div>

<script>
async function loadCommands() {
  const response = await fetch('/api/getCommand');
  const commands = await response.json();
  commands.forEach(k => addCommand(k));
}

function addCommand(k) {
  const cmd = document.createElement('div');
  cmd.innerHTML = k.command;
  cmd.setAttribute('draggable', true);
  document.getElementById(k.category).appendChild(cmd);
}
```

```
function allowDrop(ev) { ev.preventDefault(); }
function drop(ev) {
  ev.preventDefault();
  const data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getEle-
mentById(data).cloneNode(true));
}

document.addEventListener('DOMContentLoaded', loadCommands);
</script>
```

Abbildung 18 Beispiel wie Befehle aus einer Datenbank ausgelesen werden können

Um sicherzustellen, dass am Ende ein gültiges Robot-File entsteht, muss im Hintergrund ein vollständiges Robot-File mit den vordefinierten Befehlen vorliegen. Ein Beispiel für zwei Testfälle in valider Robot-Syntax ist in Abbildung 19 dargestellt. Dabei folgt die Struktur einem festen Schema, bei dem die Kategorie als Kommentar über dem Testfall angegeben wird. Fehlt dieser Kommentar, wird der Befehl automatisch der Kategorie „Ohne Kategorie“ zugeordnet. Dieses Format ermöglicht es, zusätzliche Informationen wie Argumente oder Dokumentationen einzulesen, um die Bedienbarkeit zu verbessern und die Implementierung komplexer Abläufe zu vereinfachen.

```
# Kategorie: Kartenoperationen
Prüfe Karte Erkannt
    [Arguments]    ${kartentyp}
    [Documentation]    Prüft, ob eine angegebene Karte korrekt vom
Kartenleser erkannt wird und gibt den Status zurück. Mögliche Werte
für ${kartentyp}: `Ordinationskarte`, `Gesundheitskarte`, 'Fehlerfall-
karte'.
    Log    Prüfe, ob ${kartentyp} korrekt erkannt wurde
    ${erkannt} =    Run Command And Return Status    Lese Karte Aus
${kartentyp}
    Should Be True    ${erkannt}    ${kartentyp} wurde nicht korrekt
erkannt

# Kategorie: Geräte-Management
Starte Kartenleser Neu
    [Documentation]    Startet den Kartenleser neu, um das System in
den Grundzustand zu versetzen.
    Log    Starte den Kartenleser neu
    ${status} =    Run Process    python3    ${CURDIR}/script.py
    Should Be Equal As Strings    ${status.stdout}    Hallo von
script.py!
```

Abbildung 19 Beispielkonzept für die Struktur des zugrunde liegenden Datenquelle in valider Robot-Syntax

Im Menü unter "Aktionen" besteht die Möglichkeit, neue Befehle anzulegen. Die Idee dahinter ist, dass ein neuer Befehl in der Datenbank gespeichert wird, der direkt von Fachexpert:innen auf der Plattform verwendet werden kann. Damit dieser jedoch korrekt ausgeführt werden kann, muss es zunächst implementiert werden. Hierzu wird automatisch ein Ticket im bestehenden Ticketsystem erstellt, wodurch das Entwicklerteam benachrichtigt wird. Sobald der neue Befehl umgesetzt wurde, erhalten auch die Fachexpert:innen eine automatische Benachrichtigung.

Sobald ein Testfall visuell erstellt ist, kann er auf einer Pipeline ausgeführt werden, die den Test auf dem Roboterarm automatisiert. Die Plattform ermöglicht es, den Testfall in einer vorbereiteten Pipeline zu exportieren und auszuführen. Im folgenden Code-Snippet (Abbildung 20) wird der Prozess dargestellt, wie ein Testfall in die Pipeline exportiert und in einem zentralen Speicher abgelegt wird. Dies beinhaltet mehrere Schritte, darunter das Codieren der Datei, das Entfernen von Zeilenumbrüchen und das Hochladen der codierten Datei in ein GitHub-Repository (in diesem Fall) mit einem zufällig generierten Dateinamen. Diese Automatisierung stellt sicher, dass Testfälle systematisch und nachvollziehbar gespeichert werden.

```
#!/bin/bash

# Base64-kodiere die Datei
base64 -i test.robot -o encoded_robotfile.txt

# Entferne Zeilenumbrüche, falls nötig
cat encoded_robotfile.txt | tr -d '\n' > encoded_robotfile_clean.txt

# Zufälliger Dateiname basierend auf dem aktuellen Timestamp
FILENAME="robotfile_$(date +%Y%m%d%H%M%S).robot"

# Base64-Inhalt laden
CONTENT=$(cat encoded_robotfile_clean.txt)

# GitHub Token hinzufügen
TOKEN="ghp_XXXX"

# Datei mit zufälligem Namen hochladen
curl -X PUT \
  -H "Authorization: token $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "message": "Add new robot file with random name",
    "content": "'"$CONTENT"'",
    "branch": "test-branch"
  }' \
  "https://api.github.com/repos/xxx/robodrop/contents/$FILENAME"
```

Abbildung 20 Beispiel, wie auf der Plattform erstellte Testfälle an eine bestehende CI-Pipeline geschickt werden könnten

Die Testfälle werden in ein maschinenlesbares Format konvertiert und über eine Schnittstelle hochgeladen. Die Datei wird in einem Repository gespeichert und dient als Grundlage für die Ausführung der Testfälle über die CI-Pipeline.

Während der Testausführung zeigt das Dashboard den Fortschritt der Tests an. Die Statusinformationen werden farbcodiert angezeigt, sodass Benutzer:innen sofort erkennen können, welche Testfälle erfolgreich waren, welche in der Ausführung sind und welche fehlgeschlagen sind, wie in Abbildung 21 zu sehen ist.

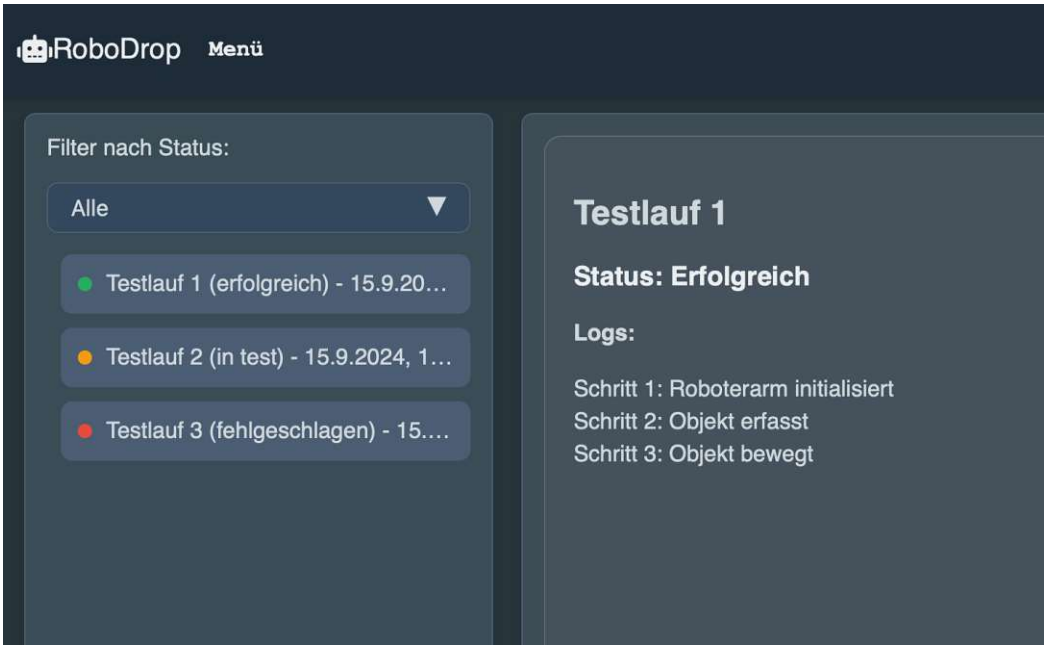

Abbildung 21 Dashboard mit Testergebnissen

Im integrierten Wiki dokumentieren Entwickler:innen und Fachexpert:innen Details zu Befehlen und zu den Funktionen der Plattform. Der Prototyp der No-Code Plattform im Rahmen dieser Arbeit bietet die Möglichkeit, diese Seiten zu kommentieren. Kommentare können direkt hinzugefügt werden, was den Wissensaustausch fördert und die Zusammenarbeit erleichtert. Abbildung 22 veranschaulicht eine mögliche Implementierung eines solchen Wikis.

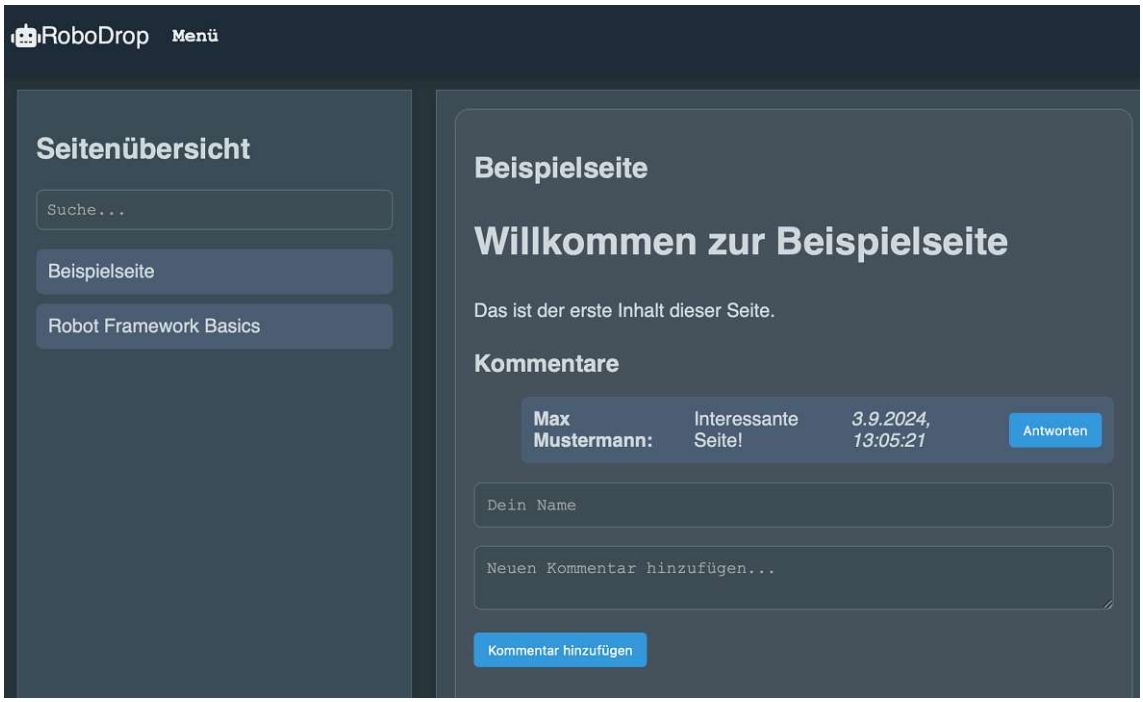

Abbildung 22 Wiki mit Kommentarfunktion

Durch die Kommentarfunktion können Benutzer:innen Feedback zu Testfällen hinterlassen oder Fehler detailliert beschreiben. Diese Funktion erleichtert die Zusammenarbeit und sorgt dafür, dass wichtige Informationen zum Testprozess dokumentiert werden.

Zusammengefasst bietet der Prototyp eine Lösung für die Erstellung, Verwaltung und Ausführung von Testfällen in einer No-Code-Umgebung. Von der Drag-and-Drop-Funktionalität für die Testfallerstellung über die Verwaltung von Befehlen bis hin zur Überwachung der Testausführung im Dashboard deckt die Plattform alle wesentlichen Funktionen ab. Durch die klare Trennung von Testfallerstellung und -überwachung wird der Workflow für die Nutzer optimiert, während die technischen Details im Hintergrund verwaltet werden. Der Prototyp wurde so konzipiert, dass er nahtlos in bestehende CI/CD-Systeme integriert werden kann, wie im nächsten Abschnitt beschrieben.

6.4 Integration in CI/CD-Pipelines

Im Rahmen dieses Projekts wird die nahtlose Integration der No-Code-Plattform in Continuous Integration/Continuous Delivery (CI/CD)-Systeme ermöglicht. Diese Integration erlaubt die automatische Ausführung von Testfällen, ohne dass manuelle Eingriffe erforderlich sind. Durch die Einbindung in gängige CI/CD-Pipelines können die Testfälle kontinuierlich ausgeführt und ihre Ergebnisse automatisch ausgewertet werden. Dies trägt wesentlich zur Effizienz und Qualitätssicherung bei, da Änderungen sofort auf ihre Funktionalität hin überprüft werden, ohne den Entwicklungsprozess zu beeinträchtigen (siehe 2.9).

Ein typischer Ablauf beginnt mit der Einrichtung eines separaten Branches in der Versionsverwaltung, der ausschließlich der Verwaltung und Ausführung von Testfällen dient (erklärt in 2.9). Dieser Test-Branch wird genutzt, um neue oder geänderte Testfälle hochzuladen. Sobald ein Commit in diesen Branch vorgenommen wird, startet die Pipeline automatisch den Testprozess. Diese Trennung der Branches gewährleistet, dass die Entwicklung nicht durch die Testläufe beeinflusst wird. Ein Beispiel für die Integration in eine bestehende Pipeline ist in Abbildung 23 zu sehen:

```
name: Run Tests and Analyze Logs
on:
  push:
    branches:
      - main
      - test-branch

jobs:
  test-and-analyze:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v4
        with:
          python-version: '3.x'
      - run: pip install -r requirements.txt && pip install robot-
framework
```

```
      - run: export PYTHONPATH=$PYTHONPATH:$(pwd)/tests && robot --
loglevel DEBUG --outputdir results tests/ || true
      - run: cat results/output.xml
      - run: python analyze_logs.py results/output.xml
      - run: cat error_report.txt
      - if: failure()
        run: echo "::error file=failed_tests_report.txt::$(cat
failed_tests_report.txt)"
      - uses: actions/upload-artifact@v4
        with:
          name: robot-logs
          path: results/
```

Abbildung 23 Beispiel-Konfiguration einer Pipeline, um Tests per Knopfdruck starten zu können

Dieser Code definiert eine vollständige CI/CD-Pipeline, die den Prozess der Testautomatisierung steuert. Die Pipeline führt verschiedene Schritte durch, um sicherzustellen, dass die Testumgebung bereitgestellt, die Tests ausgeführt und die Ergebnisse analysiert werden. Zu Beginn wird das Repository aus dem Test-Branch ausgecheckt, und die Umgebung wird mit Python 3.x eingerichtet. Anschließend werden die notwendigen Abhängigkeiten installiert, darunter auch Bibliotheken wie das Robot Framework, die für die Ausführung der Tests mit dem entwickelten Prototyp erforderlich sind.

Der nächste Schritt ist die Ausführung der Testfälle, die im Test-Branch hinterlegt sind. Dabei werden alle Testschritte protokolliert, und das Robot Framework erstellt eine detaillierte Log-Datei (`results/output.xml`), die alle Aktionen des Roboterarms sowie die entsprechenden Testergebnisse enthält. Nach der Erstellung der Log-Datei erfolgt deren Analyse mittels des eigens entwickelten Python-Skripts (`analyze_logs.py`), um zu überprüfen, ob alle Tests erfolgreich verlaufen sind oder Fehler aufgetreten sind (siehe Kapitel 6.6). Sollte ein Test fehlschlagen, wird ein Fehlerbericht generiert, der detaillierte Informationen zum Problem liefert und dem Nutzer eine schnelle Identifikation der Ursache ermöglicht.

Zuletzt werden alle generierten Logs gespeichert, sodass sie an die No-Code-Plattform übermittelt werden können. Dieser Prozess stellt sicher, dass die Testausführung vollständig automatisiert abläuft und dass alle Testergebnisse und Logs sofort verfügbar sind, ohne dass manuelle Eingriffe erforderlich sind.

Der wesentliche Vorteil dieser Integration in CI/CD-Systeme besteht darin, dass die Testfälle automatisiert und unabhängig vom eigentlichen Entwicklungsprozess ausgeführt werden können. Dabei werden die Testfälle in einem separaten Branch verwaltet, sodass Entwicklerinnen und Entwickler neue Funktionen in ihren eigenen Branches weiterentwickeln können, während die Testfälle kontinuierlich im Test-Branch laufen. Dies fördert eine frühzeitige Fehlererkennung und entkoppelt den Testvorgang vom regulären Entwicklungsablauf.

Allerdings erfordert ein solches Vorgehen ein dediziertes Hardware-Setup, in welchem physische Geräte für die Testausführung bereitstehen. Da nur so viele Pipelines parallel betrieben werden können, wie es die verfügbare Hardware zulässt, führt eine begrenzte Kapazität schnell zu Engpässen. Darüber hinaus könnten Ausfälle in der Testhardware schwerwiegende Folgen nach sich ziehen, indem sie sämtliche laufenden Testprozesse

unterbrechen oder verzögern. In vielen Fällen ist deshalb eine entsprechende Überwachung und Wartung der Testumgebung erforderlich, um Ausfallzeiten zu minimieren und die Zuverlässigkeit der Tests sicherzustellen. Eine Betrachtung dieser Herausforderungen sowie möglicher Maßnahmen zur Risikominimierung erfolgt im folgenden Abschnitt.

6.5 Herausforderungen und Risikominimierung in hardwaregestützten Testumgebungen

Die Einbindung dedizierter Hardware in automatisierte CI/CD-Pipelines ermöglicht eine realitätsnahe Abbildung produktiver Umgebungen und eignet sich besonders für Tests, bei denen rein virtuelle oder emulierte Lösungen die zugrunde liegende Komplexität nicht vollständig widerspiegeln können. Allerdings gehen damit diverse Herausforderungen einher. Eine wesentliche Schwierigkeit liegt in der Abhängigkeit von physischen Ressourcen, die in ihrer Anzahl begrenzt sind und zudem regelmäßiger Wartung, Aktualisierung und Überwachung bedürfen. Fällt eine zentrale Hardwarekomponente aus, kann dies sämtliche laufenden Testprozesse unterbrechen und für Verzögerungen im Entwicklungsablauf sorgen. Engpässe entstehen insbesondere dann, wenn nur eine begrenzte Menge an Geräten zur Verfügung steht und mehrere Projekte parallel Testkapazitäten benötigen.

Um diesen Risiken zu begegnen, bieten sich unterschiedliche Maßnahmen an. Eine sorgfältige Skalierung der Hardwareausstattung ist essenziell, damit auch bei Lastspitzen genügend Kapazitäten verfügbar sind. Redundanz in Form mehrerer identischer Geräte oder Teststationen kann verhindern, dass der Ausfall einzelner Komponenten zum Stillstand aller Pipeline-Läufe führt. Darüber hinaus ist es empfehlenswert, Backup- und Notfallkonzepte zu entwickeln, damit bei Komplettausfällen schnell auf alternative Systeme umgestellt werden kann. Ein engmaschiges Monitoring mithilfe entsprechender Tools ermöglicht das frühzeitige Erkennen von Problemen, beispielsweise Überhitzung oder sich anbahnende Hardwaredefekte, was wiederum präventive Wartung erleichtert.

Insgesamt bietet der Einsatz dedizierter Hardware zwar den Vorteil einer besonders realitätsnahen Testumgebung, setzt jedoch eine gewissenhafte Planung und angemessene Ressourcen voraus. Eine durchdachte Infrastruktur mit ausreichender Kapazität, redundanten Komponenten, wirksamen Backup-Strategien und kontinuierlichem Monitoring verringert die Gefahr, dass ein einzelner Hardwarefehler den gesamten Testbetrieb lahmlegt. Gleichzeitig stärkt eine solche Herangehensweise die Stabilität und Zuverlässigkeit des Entwicklungs- und Release-Prozesses, da potenzielle Probleme frühzeitig erkannt und effektiv adressiert werden können.

6.6 Fehlererkennung und Log-Analyse

Die Fehlererkennung und Log-Analyse sind entscheidend, um auftretende Probleme in den Testfällen oder bei der Steuerung des Roboterarms frühzeitig zu identifizieren und zu beheben. Durch die Automatisierung dieser Prozesse wird sichergestellt, dass Fehler systematisch erfasst und den entsprechenden Kategorien zugeordnet werden können, ohne dass manuelle Eingriffe nötig sind. Sobald ein Testlauf über die Plattform und die angebundene CI/CD-Pipeline gestartet wird, beginnt die Plattform automatisch mit der

Erfassung der Log-Daten. Diese Logs enthalten alle relevanten Informationen über den Testablauf, einschließlich der ausgeführten Befehle, der Bewegungen des Roboterarms und eventueller Rückmeldungen oder Fehlermeldungen. Die Logs werden direkt aus der Pipeline abgerufen und in einem strukturierten Format gespeichert. Entwickler:innen haben die Möglichkeit, das genaue Format der Logs selbst festzulegen, sodass sie den spezifischen Anforderungen des Projekts angepasst werden können. Wichtig ist dabei, dass die Logs kompatibel mit dem Analyse-Skript der Plattform sind, welches die Daten nach festgelegten Kriterien analysiert.

Das Analyse-Skript übernimmt die automatische Überprüfung der generierten Logs und wird vom Entwicklerteam bereitgestellt. Dabei sucht es nach vordefinierten Mustern oder Schlüsselwörtern, die auf Fehler hinweisen könnten. Dies könnten zum Beispiel Fehlermeldungen sein, die darauf hindeuten, dass der Roboterarm ein Hardwareproblem hat oder falsche Parameter übergeben wurden. Um eine präzise Fehlererkennung zu gewährleisten, ist es wichtig, dass die Entwickler:innen beim Erstellen der Testbefehle eine effektive Fehlerbehandlung implementieren. Dies bedeutet, dass bei jedem möglichen Fehlerfall detaillierte Informationen in den Logs hinterlassen werden, die es dem Analyse-Skript ermöglichen, den Fehler eindeutig zu identifizieren und zu klassifizieren.

Das folgende Skript zeigt, wie eine vereinfachte Log-Analyse umgesetzt werden könnte Abbildung 24:

```
def lese_logs(xml_datei):
    # XML-Datei parsen und Testfälle extrahieren
    # Jedes Testfall enthält:
    # - ID
    # - Name
    # - Status (Pass oder Fail)
    # - Schritte mit jeweiligen Status und Lognachrichten
    return test_daten

def kategorisiere_fehler(log_nachricht):
    # Fehlerkategorien erkennen, z.B. Datenbankfehler, Netzwerkfehler
    # Anhand von Schlüsselwörtern in der Lognachricht
    return fehler_kategorie

def übersetze_und_vereinfache_nachricht(nachricht):
    # Technische Fehlermeldungen in lesbare Sprache übersetzen und
vereinfachen
    return vereinfachte_nachricht

def erstelle_bericht(test_daten):
    bericht = []
    for testfall in test_daten:
        bericht.append(f"Test Case ID: {testfall['id']}")
        bericht.append(f"Test Name: {testfall['name']}")

        # Status des Tests (Erfolgreich oder Fehlgeschlagen)
        test_status = "Erfolgreich" if testfall['status'] == 'PASS'
else "Fehlgeschlagen"
        bericht.append(f"Test Status: {test_status}")

        # Schritte innerhalb des Tests
        for schritt in testfall['schritte']:
```

```
            schritt_status = "Erfolgreich" if schritt['status'] ==
'PASS' else "Fehlgeschlagen"
            bericht.append(f"  - Schritt: {schritt['name']}")
            bericht.append(f"    Status: {schritt_status}")

            # Falls fehlgeschlagen, Fehler kategorisieren und Nach-
richt anzeigen
            if schritt_status == "Fehlgeschlagen":
                fehler_kategorie = kategorisiere_fehler("
".join(schritt['nachrichten']))
                bericht.append(f"    Fehlerkategorie: {fehler_katego-
rie}")
                vereinfachte_nachricht = übersetze_und_vereinfa-
che_nachricht(schritt['nachrichten'][0])
                bericht.append(f"    Detaillierter technischer Fehler:
{vereinfachte_nachricht}")

    return "\n".join(bericht)

def speichere_bericht(bericht, dateiname):
    with open(dateiname, 'w') as datei:
        datei.write(bericht)

# Hauptablauf
test_daten = lese_logs("logs.xml")
bericht = erstelle_bericht(test_daten)
speichere_bericht(bericht, "bericht.txt")
```

Abbildung 24 Beispiel-Implementierung für einen simplen Loganalyzer

Dieses Skript liest die Logs aus einer XML-Datei, analysiert sie auf Basis vordefinierter Fehlerkategorien und erstellt einen detaillierten Fehlerbericht. Das Skript erkennt Muster in den Dateien wie Netzwerkfehler oder Anwendungsfehler und kategorisiert sie entsprechend. Die Ergebnisse werden in einem leicht verständlichen Bericht zusammengefasst, der die Testschritte und eventuelle Fehler beschreibt.

Die Plattform unterstützt mehrere Kategorien von Fehlern, im Vorfeld definiert und regelmäßig aktualisiert werden können. Dazu gehören Hardwareprobleme, wie etwa Verbindungsfehler mit dem Roboterarm, Anwendungsfehler, die durch fehlerhafte Parameter in den Testfällen verursacht werden, und allgemeine logische Fehler, die auf Probleme im Testablauf hinweisen. Die automatisch erkannten Fehler werden im Dashboard der Plattform angezeigt, sodass die Nutzer sofort über etwaige Probleme informiert werden.

Sobald ein Fehler erkannt wird, zeigt das Dashboard eine entsprechende Warnung an, und der fehlerhafte Testlauf wird farblich hervorgehoben. Dies ermöglicht es den Nutzern, auf einen Blick zu erkennen, welche Testfälle erfolgreich abgeschlossen wurden und bei welchen es Probleme gab. Durch die farbliche Codierung, wie rot für fehlgeschlagene und grün für erfolgreiche Testläufe wird die Übersichtlichkeit gewährleistet. Nutzer können durch einen Klick auf den fehlerhaften Testlauf vereinfachte Informationen abrufen, einschließlich der Log-Dateien, die den genauen Ablauf des Tests in Schritten dokumentieren.

Die Log-Analyse ist darauf ausgelegt, die Fehlerbehebung so effizient wie möglich zu gestalten. Zusätzlich zur Anzeige im Dashboard können die generierten Fehlerberichte automatisch in Form von Jira-Tickets oder anderen Aufgabenmanagement-Tools bereitgestellt werden. Diese Tickets enthalten bereits alle relevanten Informationen, wie die genaue Fehlerkategorie, die zugehörigen Log-Daten und die verantwortlichen Testfälle. Dadurch wird sichergestellt, dass die Fehler direkt dem zuständigen Entwicklerteam zugewiesen werden und alle notwendigen Informationen zur Behebung des Problems verfügbar sind. Dies beschleunigt nicht nur den Fehlerbehebungsprozess, sondern erhöht auch die Nachverfolgbarkeit der aufgetretenen Probleme.

Insgesamt sorgen die automatisierte Fehlererkennung und Log-Analyse der Plattform dafür, dass potenzielle Probleme frühzeitig erkannt, klassifiziert und den Entwickler:innen zur Verfügung gestellt werden. Die klare Struktur der Log-Daten und die automatisierte Verarbeitung durch das Analyse-Skript ermöglichen eine schnelle Identifikation der Fehlerursachen, was die Zuverlässigkeit und Effizienz der Testprozesse deutlich erhöht. Ergänzend dazu wird bei Fehlern, die vom Skript nicht interpretiert werden können, der ursprüngliche Log-Eintrag unverändert an die Entwickler:innen weitergeleitet. Auf diese Weise gehen keine Informationen verloren, und auch unvorhergesehene oder komplexe Fehlerfälle können manuell analysiert und in den kontinuierlichen Verbesserungsprozess der Plattform integriert werden.

6.7 Erweiterbarkeit der prototypischen Umsetzung des Konzepts

Die Plattform wurde so gestaltet, dass sie flexibel und erweiterbar ist, um den unterschiedlichen Bedürfnissen von Entwickler:innen und Fachexpert:innen gerecht zu werden. Ein zentraler Fokus liegt auf dem No-Code-Konzept, das es Fachexpert:innen ohne Programmierkenntnisse ermöglicht, Testfälle zu erstellen und Roboterarme zu automatisieren, ohne sich mit technischen Details auseinandersetzen zu müssen.

Entwickler:innen können neue Befehle und Funktionen hinzufügen, um sie an spezifische Anforderungen anzupassen. Das zugrunde liegende Robot Framework, ein etabliertes Framework für Testautomatisierung, unterstützt diese Erweiterbarkeit ideal, da es auf einer modularen und flexiblen Architektur basiert.

Das Robot Framework bietet zahlreiche Bibliotheken und Funktionen, die für unterschiedliche Automatisierungsszenarien nützlich sind. So lassen sich etwa Bibliotheken wie Selenium[25] für Webautomatisierung, Appium[26] für Automatisierungen von mobilen Apps oder auch CnCLibrary[27] für eine direkte Maschinensteuerung leicht integrieren. Durch die Verwendung eines Low-Code Frameworks sparen die Entwickler:innen bei der Implementierung der Befehle viel Zeit und erzeugen dabei auch sehr wenig Code, was die Wartung vereinfachen kann. Falls spezifische Anforderungen bestehen, können

[25] https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html
[26] https://docs.robotframework.org/docs/different_libraries/appium
[27] https://github.com/eficode/robotframework-cnclibrary

Entwickler:innen auch eigene Funktionen in Python schreiben und diese nahtlos integrieren, was die Plattform sowohl für einfache als auch komplexe Anwendungen skalierbar macht [129].

Die Befehle für die Testfallerstellung werden zentral verwaltet, beispielsweise in einer Datenbank oder in einer zentralen Datei, die in einem Feature-Branch abgelegt ist. Diese zentrale Verwaltung ermöglicht es dem Entwicklerteam, die Befehle jederzeit zu erweitern oder zu aktualisieren. Dadurch kann die Plattform flexibel an neue Anforderungen angepasst und kontinuierlich verbessert werden, während sie dynamisch und aktuell bleibt.

Insgesamt bietet die Plattform eine Kombination aus No-Code-Bedienung für Fachpersonen und umfassender Erweiterbarkeit für Entwickler:innen. Durch die Integration des Robot Frameworks und die Möglichkeit, eigene Funktionen hinzuzufügen, wird eine hohe Flexibilität erreicht, die sowohl für einfache als auch für komplexe Automatisierungsszenarien geeignet ist.

7 Ergebnisse

Im nachfolgenden Kapitel werden die Resultate aus den durchgeführten Expert:inneninterviews vorgestellt und interpretiert. Der Fragebogen sowie die vollständigen Transkripte sind in Kapitel 11.4 beziehungsweise 11.5 einsehbar. Abschnitt 7.2 widmet sich zunächst der quantitativen Auswertung, bei der statistische Zusammenhänge und Tendenzen anhand von Prozentangaben illustriert werden. In Abschnitt 7.3 stehen die qualitativ erhobenen Erkenntnisse aus den offenen Fragen und Gesprächsbeiträgen im Vordergrund, ehe in 7.4 eine zusammenfassende Bewertung aller gewonnenen Ergebnisse erfolgt.

7.1 Methodik

Die durchgeführten Expert:inneninterviews wurden so angelegt, dass die Teilnehmenden einen Einblick in das vorgestellte Konzept und seine potenzielle Anwendung gewinnen konnten [130]. Zu diesem Zweck wurden zunächst fünf Personen mit langjähriger Erfahrung in unterschiedlichen IT-Bereichen (z. B. Testmanagement, Softwareentwicklung, IT-Architektur, Projektleitung) ausgewählt. Diese Auswahl gewährleistete ein möglichst breites Meinungsspektrum, da die Befragten durch ihre unterschiedlichen beruflichen Hintergründe und Rollen verschiedene Perspektiven auf das zu untersuchende Konzept einbringen konnten [126].

Bevor die eigentlichen Fragen gestellt wurden, erhielten alle Teilnehmenden eine kurze Vorstellung des Ziels: Fachexpert:innen sollen mithilfe des Konzepts einfacher in den Testprozess eingebunden werden, um Tests ohne umfangreiche Programmierkenntnisse erstellen zu können. Im Anschluss daran wurden drei voneinander unabhängige Szenarien, die exemplarisch verdeutlichten, wie die Plattform in einem Smart-Schloss-Projekt, einer NFC-basierten Handy-App und einem Gesundheitsprojekt mit Kartenleser eingesetzt werden kann. Dieser Demonstrationsteil diente dazu, die Rollenverteilung (Fachexpert:in vs. Entwickler:in), den Aufbau eines Testschritts und die Rückmeldeschleifen innerhalb einer CI-Pipeline sichtbar zu machen. Die Expert:innen konnten diese Schritte live verfolgen und wurden dazu angeregt, während der Vorführung Fragen zu stellen oder spontane Anmerkungen zu notieren, um etwaige Unklarheiten sofort anzusprechen.

Direkt im Anschluss an die Demonstration kamen die vorbereiteten Interviewfragen (siehe 11.4) zum Einsatz, der sowohl geschlossene Fragen als auch offene, explorative

Elemente enthielt. Die geschlossenen Fragen in Form einer Skala von 1 („stimme überhaupt nicht zu“) bis 4 („stimme voll zu“), halfen dabei, die grundsätzliche Zustimmung zu zentralen Themen wie No-Code-Ansatz, CI-Integration und Rollenverteilung statistisch zu erfassen [48]. Es wurden sowohl quantitative (geschlossene) als auch qualitative (offene) Fragen gestellt, um einerseits die relevanten Themenbereiche systematisch abzudecken und andererseits tiefgehende Diskussionen anzuregen, die unerwartete Einsichten und Perspektiven ermöglichen [125]. Die offenen Fragen ermöglichten es den Expert:innen, detaillierte Einschätzungen zu geben, wie das Konzept in realen Projekten wirken könnte, wo sie die größten Vorteile oder potenziellen Risiken sehen und welche konkreten Voraussetzungen erfüllt sein müssen, damit es erfolgreich eingesetzt werden kann [10][66]. Um eine realistische Einschätzung zu erzielen, wurde bei unklaren oder kontroversen Aussagen nachgefragt, sodass sich eine lebhafte Diskussion entwickelte, in der die Teilnehmenden auf organisatorische Aspekte (z. B. Terminologie-Konzept, Review-Prozesse, Rollenmodell) und technische Anforderungen (z. B. Versionsverwaltung, Duplikate, Wartbarkeit) eingingen.

Alle Interviews wurden mit Zustimmung der Teilnehmenden aufgezeichnet und anschließend transkribiert. Diese Transkriptionen bildeten die Grundlage für die spätere Auswertung. Im Rahmen der quantitativen Analyse wurden die Skalenantworten zusammengefasst, um eine erste Einschätzung zur Verteilung der Zustimmung oder Skepsis gegenüber bestimmten Aussagen zu gewinnen. Darauf aufbauend erfolgte eine qualitative Analyse, bei der wiederkehrende Themen, kontroverse Punkte und konkrete Verbesserungsvorschläge aus den Gesprächsprotokollen herausgearbeitet wurden [130]. In diesem Schritt wurden auch die spontanen Anmerkungen der Expert:innen während der Demonstrationsphasen einbezogen, um ein möglichst vollständiges Bild ihrer Überlegungen und Bedenken zu erhalten.

Zusätzlich bekamen die Expert:innen im Nachgang eine kompakte Zusammenfassung ihrer Kernaussagen und hatten die Möglichkeit, Unstimmigkeiten zu korrigieren oder weitere Ergänzungen vorzunehmen. Diese Validierungsschleife sollte sicherstellen, dass individuelle Aussagen inhaltlich richtig erfasst worden waren und sowohl die positiven als auch die kritischen Stimmen in die Auswertung einflossen [28]. Auf diese Weise konnte das Vorgehen sowohl auf methodischer Ebene (standardisierte Skalenfragen plus ausführliche Transkription) als auch auf inhaltlicher Ebene (Mischung aus praktischer Demonstration und reflexiven Diskussionsfragen) möglichst zuverlässig und nachvollziehbar gestaltet werden.

7.2 Quantitative Analyse

Zu Beginn wurden demografische Fragen gestellt [48]. Wie Abbildung 25 dargestellt, befinden sich 40 % der Teilnehmenden im Alterssegment zwischen 30 und 39 Jahren, während 60 % der Befragten der Altersgruppe 40 bis 50 zuzuordnen sind.

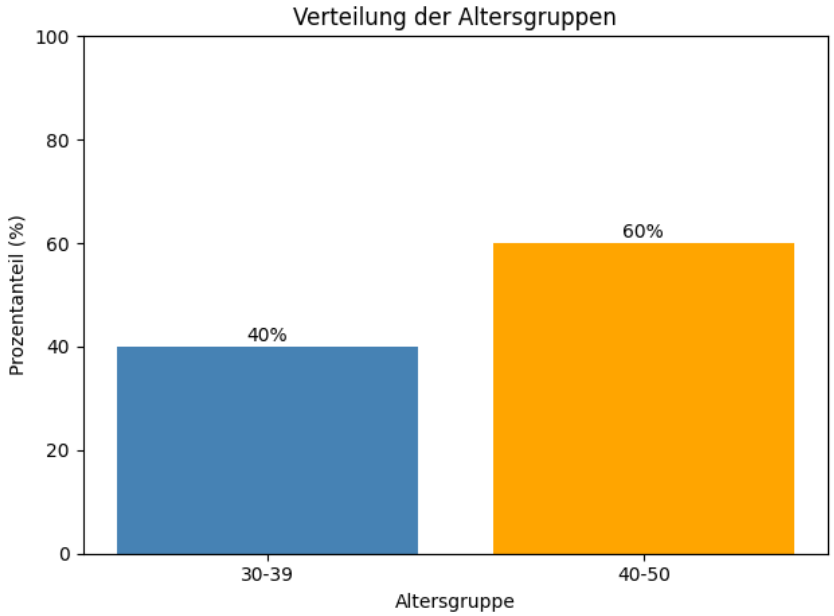

Abbildung 25 Berufserfahrung der Befragten in IT und im Fachgebiet

Betrachtet man die Gesamterfahrung im IT-Bereich, so verfügen 80 % der Befragten über mehr als zehn Jahre Berufspraxis (Abbildung 26). Im Hinblick auf die fachspezifische Erfahrung (z. B. im Testmanagement, in der Softwareentwicklung oder in der IT-

Architektur) zeigt sich, dass die Mehrheit (60 %) zwischen fünf und 15 Jahren Branchenerfahrung angibt.

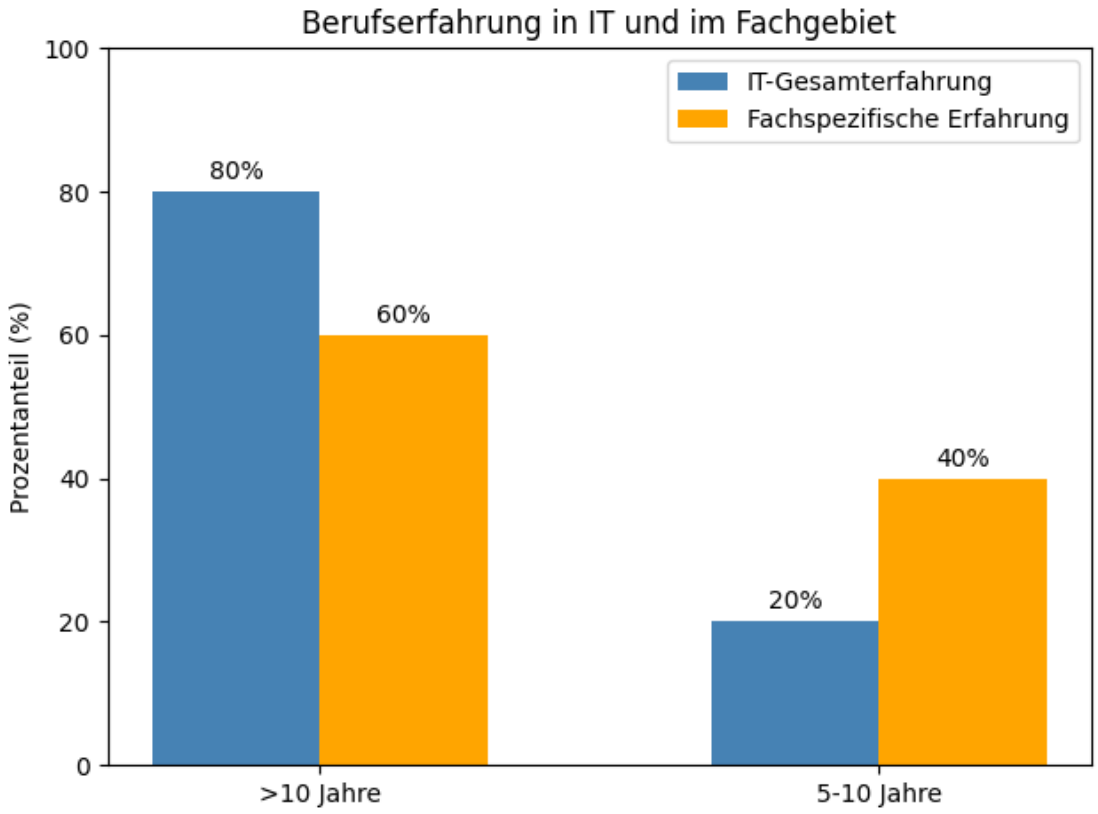

Abbildung 26 Angegebene Ausbildung der Befragten

Bezüglich Ausbildung bzw. IT-spezifischer Qualifikation zeigt sich, dass alle Befragten mindestens eine akademische Ausbildung absolviert haben (100%). Rund 60 % vereinen mehrere Ausbildungswege, zum Beispiel einen akademischen Abschluss plus zusätzliche Zertifikate oder Kurse. Ein Balkendiagramm wie in Abbildung 27 veranschaulicht, wie oft eine Kombination verschiedener Weiterbildungen genannt wurde.

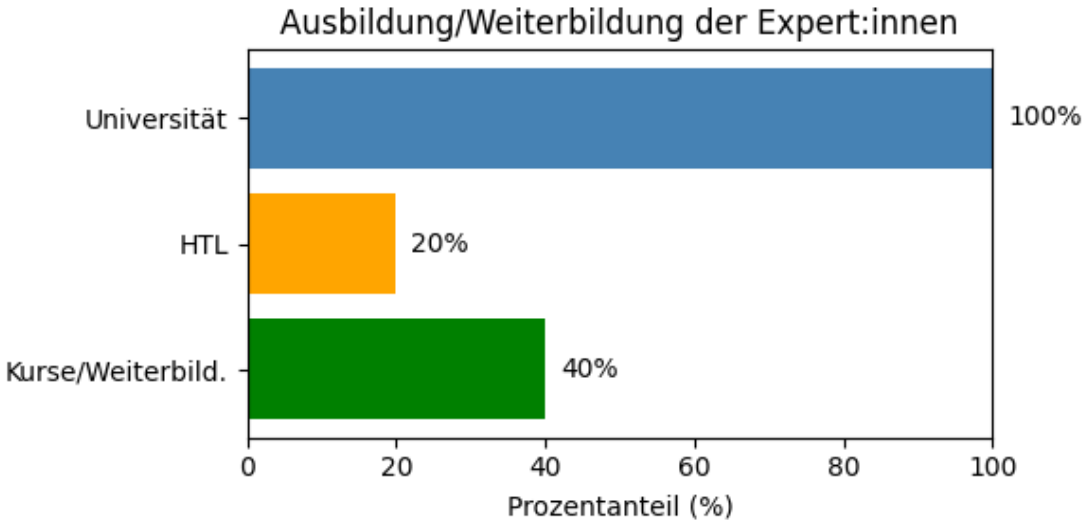

Abbildung 27 Bevorzugter Einsatzzeitpunkt des Konzeptes im Entwicklungsprozess

Auf die Frage, in welcher Phase des Entwicklungsprozesses das Konzept sinnvoll eingesetzt werden könne, gaben nahezu 100 % an, dass sie eine Einbindung in allen Phasen (von der Anforderungsanalyse bis zur Wartung) befürworten. Etwa 80 % betonten dabei die besondere Relevanz einer möglichst frühen Einbindung, um Abnahmekriterien und Use-Cases bereits in der Planungs- oder Designphase zu definieren; ein kleinerer Teil (20 %) hob zusätzlich den Nutzen für spätere Wartungs- und Regressionstests hervor (siehe Abbildung 28).

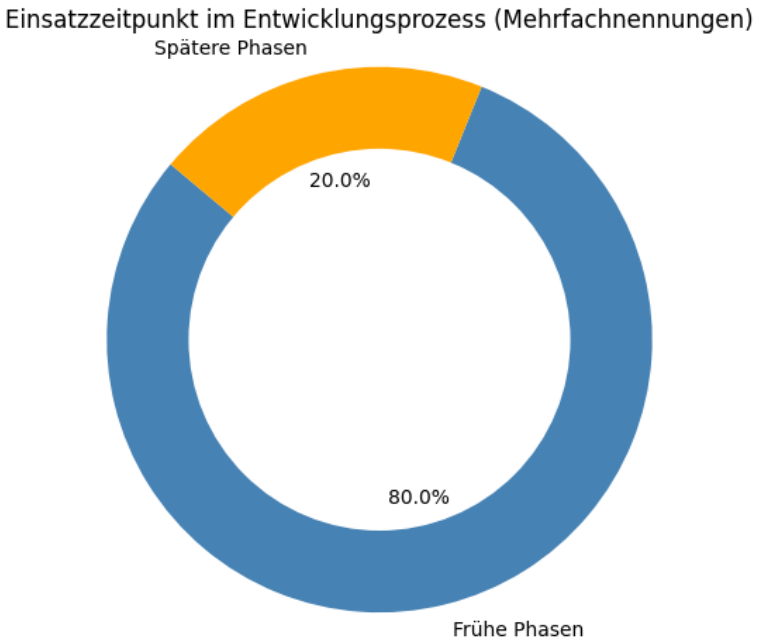

Abbildung 28 Bevorzugter Einsatzzeitpunkt des Konzeptes im Entwicklungsprozess

Ein zentrales Element der quantitativen Befragung war die Einschätzung des Nutzens des vorgestellten Konzepts für den Entwicklungsprozess. Die Skala reichte von 1 („stimme überhaupt nicht zu“) bis 4 („stimme voll zu“). Hier gaben 60 % einen Wert von 3 an, und 40 % bewerteten es sogar mit der Zahl 4, was auf eine überwiegend positive Grundhaltung schließen lässt. In Abbildung 29 wird in einem Blasendiagramm veranschaulicht, wie sich die Antworten auf die verschiedenen Zustimmungsstufen verteilen.

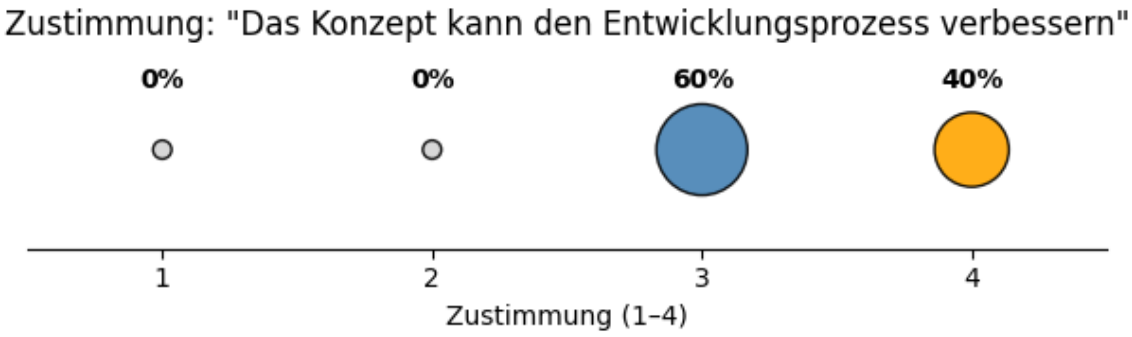

Abbildung 29 Zustimmung zum Konzept („Das Konzept kann den Entwicklungsprozess verbessern“)

Auf die Frage, wie stark das Konzept die Zusammenarbeit zwischen Fachexpert:innen und Entwicklungsteams verbessern könne, ergab sich ein ähnliches Bild: Die meisten Befragten (rund 80 %) wählten hier einen Wert von 3, ein kleinerer Teil (20 %) vergab die Höchstnote 4 (Abbildung 30). In den offenen Kommentaren wurde hierzu mehrfach betont, dass die tatsächliche Verbesserung stark von ergänzenden organisatorischen Maßnahmen, wie z. B. Review-Prozessen und klar geregelten Rollen, abhänge. So merkte ExpertIn 2 an, dass „menschliche Faktoren und die Projektorganisation“ maßgeblich seien (11.5.2). ExpertIn 1 wies auf die Notwendigkeit eines Review-Prozesses hin, indem „neue Testfälle einen ‘Draft-Status’ haben und [...] von jemandem geprüft werden, bevor sie automatisch laufen“ (11.5.1). Auch ExpertIn 5 unterstrich, dass es „eine Instanz [braucht], die kontrolliert, ob der Step und der Code wirklich übereinstimmen“ (11.5.5).

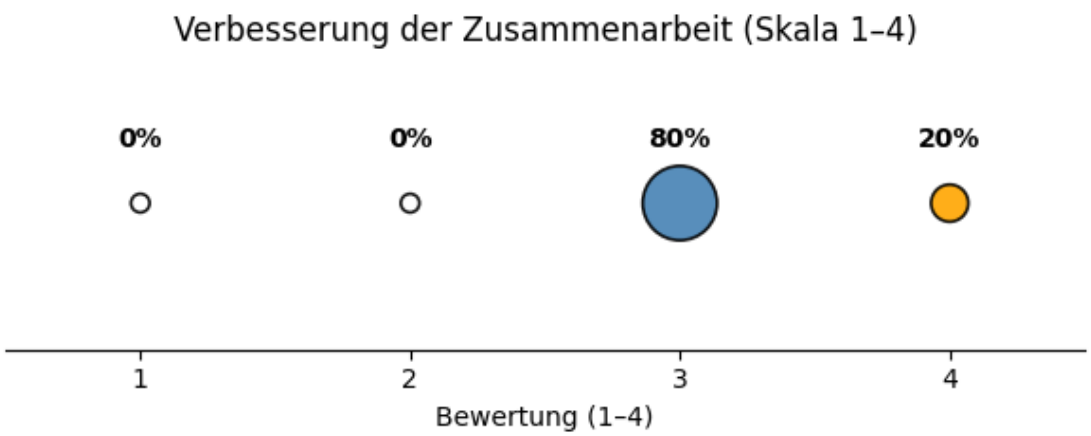

Abbildung 30 "Verbesserung der Zusammenarbeit"

Die Bedeutung der CI-Integration (siehe Kapitel 2.9) wurde von einem großen Teil (80 %) als „wichtig“ bis „sehr wichtig“ (Skalenwert 3 oder 4) eingestuft, wobei knapp 20 % der Ansicht waren, dass die CI-Einbindung für Fachexpert:innen nicht zwingend im Vordergrund stehe (Skalenwert 1 oder 2). Wie in Abbildung 31 zu sehen ist, unterstreicht die Mehrheit jedoch die Notwendigkeit einer Anbindung des Konzepts an eine CI-Pipeline.

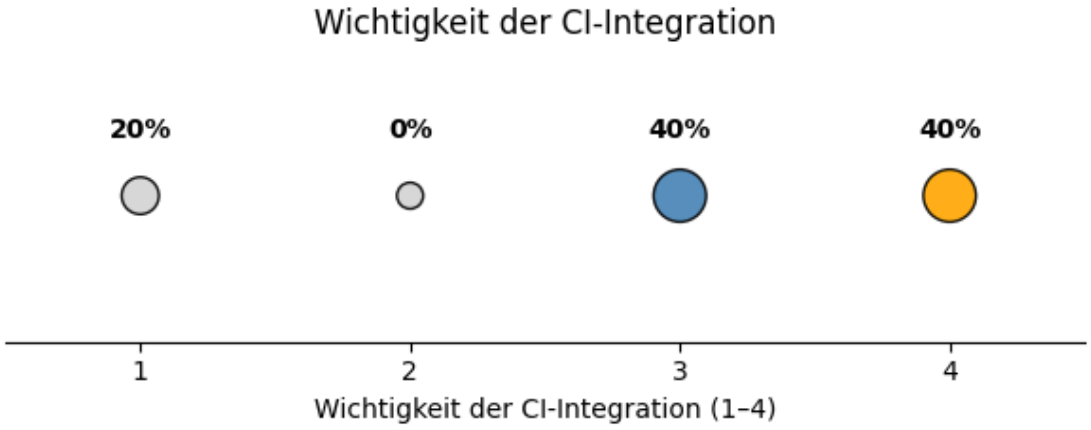

Abbildung 31 Wichtigkeit der CI-Integration

Zur Frage, wie wahrscheinlich die Teilnehmenden das Konzept künftig einsetzen würden (ebenfalls Skala 1–4), liegt die Mehrzahl (60 %) bei 3 („wahrscheinlich“), während 40 % die Höchststufe 4 („sehr wahrscheinlich“) auswählten. Auch hier zeigt die Verteilung, dass niemand den Einsatz kategorisch ausschließt. Eine Aufbereitung als Blasendiagramm, wie in Abbildung 32, veranschaulicht deutlich die optimistische Grundhaltung.

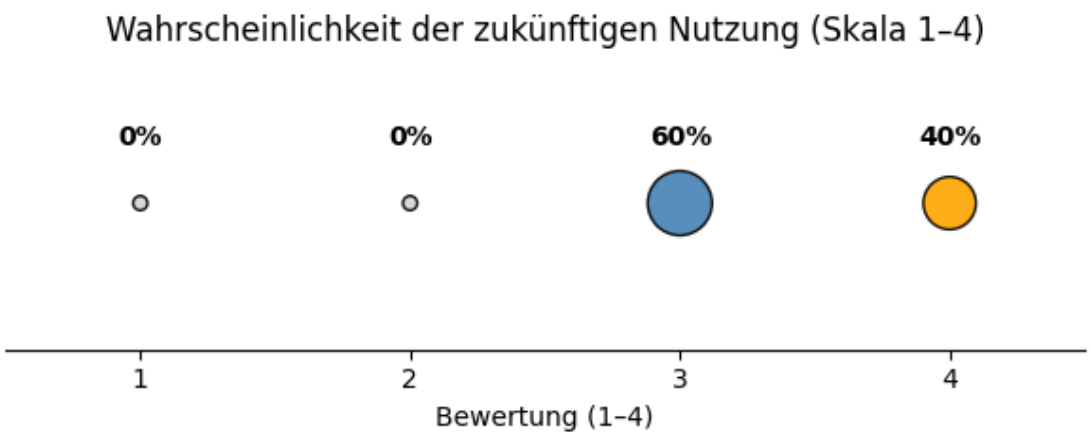

Abbildung 32 Wahrscheinlichkeit der zukünftigen Nutzung des Konzepts

Insgesamt widerspiegeln die quantitativen Daten ein überwiegend positives Meinungsbild. Die meisten Befragten stehen dem Konzept offen gegenüber und sehen insbesondere in agilen Vorgehensweisen, in frühen Anforderungsphasen sowie in der Wartungs- und Regressionstestphase einen hohen Nutzen. Zugleich wird die Integration in Continuous-Integration-Prozesse von der Mehrheit befürwortet. Aufbauend auf diesen quantitativen Tendenzen werden im nächsten Abschnitt die qualitativen Aussagen vertiefend betrachtet.

7.3 Qualitative Analyse

Die im Rahmen dieser Arbeit durchgeführten Interviews (siehe Anhang 11.5) zeigen ein breites Spektrum an Meinungen zum vorgestellten Konzept, wobei bestimmte Themen immer wiederkehrend diskutiert wurden. Um eine Übersicht zu schaffen, fasst Tabelle 1 die neun Kernaussagen zusammen und gibt zugleich Auskunft über den Anteil der Befragten (insgesamt fünf Expert:innen), die die jeweilige Aussage explizit bestätigt oder als wichtig hervorgehoben haben.

Kernaussage	Inhalt	Zustimmung in % (Anzahl)
1	Frühzeitige und kontinuierliche Einbindung der Fachexpert:innen im Entwicklungsprozess ist sinnvoll	100%
2	Das Konzept könnte die Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen verbessern	80%
3	Integration in CI/CD-Pipeline ist entscheidend für den Erfolg	80%
4	Vorteile des Konzepts – niedrigere Einstiegshürden und direkter Zugang für Fachexpert:innen	100%
5	Herausforderungen – Duplikate, Versionierung und Komplexität	60%

6	Einbindung der Fachexpert:innen in komplexen Projekten besonders wertvoll	80%
7	Notwendigkeit von Schulungen und Unterstützung für Fachexpert:innen	60%
8	Vorteile für die Testabdeckung und Qualität durch Fachwissen	40%
9	Risiken durch falsche Annahmen oder unklare Rollenverteilung	40%

Tabelle 4 Kernaussagen anhand der qualitativen Analyse der Expert:inneninterviews

Aus dieser Zusammenfassung wird ersichtlich, dass die frühzeitige Einbindung der Fachseite (Kernaussage 1) von allen Befragten befürwortet wird. Aussagen wie „Am besten so früh wie möglich, um gleich bei der Spezifikation die nötigen Testschritte anzudenken“ (ExpertIn 1) verdeutlichen, dass eine möglichst enge Kooperation zwischen Fach- und Entwicklungsteam bereits in frühen Projektphasen angestrebt wird. Ebenso einstimmig wird die niedrige Einstiegshürde (Kernaussage 4) als Vorteil angesehen: „Fachexpert:innen müssen keinen Code lesen oder schreiben. Sie klicken sich ihre Testfälle zusammen, was ihnen einen schnelleren Zugang verschafft“ (ExpertIn 2).

Während das Potenzial zur Verbesserung der Zusammenarbeit (Kernaussage 2) und die Integration in CI/CD (Kernaussage 3) jeweils von 80 % der Befragten betont werden, bestehen gewisse Vorbehalte hinsichtlich der notwendigen organisatorischen Rahmenbedingungen und menschlichen Faktoren. So heißt es bei ExpertIn 2: „Ob es die Kommunikation wirklich verbessert, hängt auch von den menschlichen Faktoren und der Projektorganisation ab.“ Auch das Einbinden der Fachseite in die technische Umsetzung von Pipeline-Läufen hält ExpertIn 3 für eher unrealistisch. Dennoch bestätigt sich die hohe Bedeutung automatisierter Prüfungen für einen durchgängigen Entwicklungsprozess (ExpertIn 4).

Die Herausforderungen (Kernaussage 5) beziehen sich vor allem auf mögliche Duplikate, unklare Terminologie und steigende Komplexität beim gemeinsamen Definieren von Testfällen. „Man braucht ein Terminologie- und Steps-Konzept, sonst degeneriert das Ganze“, warnt ExpertIn 2. Außerdem kann die Nachvollziehbarkeit zwischen Test und Code sinken, wenn die Testsprache zu stark vom Code entkoppelt wird (ExpertIn 4). 60 % der Befragten sehen hier erheblichen Handlungsbedarf, besonders wenn größere Teams und mehrere Fachexpert:innen involviert sind.

Ebenfalls auf 80 % Zustimmung stößt die Aussage, dass die Einbindung der Fachexpert:innen in komplexen Projekten (Kernaussage 6) einen besonders großen Mehrwert bietet. ExpertIn 2 fasst zusammen: „Gerade in komplexen Projekten bringt das Fachwissen enorm viel.“ Das Wissen der Domänenexpert:innen kann so gezielter eingesetzt werden und gewährleistet eine höhere inhaltliche Validität der Testfälle. In Projekten mit umfangreicher Hardware-Integration oder komplexen fachlichen Prozessen kann die frühzeitige Einbindung von Fachexpert:innen dazu beitragen, Fehler rechtzeitig aufzudecken und so den Gesamtaufwand zu verringern (vgl. ExpertIn 5).

Zugleich wird deutlich, dass eine reine Bereitstellung von Tools nicht genügt. 60 % der Befragten fordern gezielte Schulungen oder Workshops (Kernaussage 7), damit Fachex-

pert:innen den Umgang mit No-Code-Oberflächen oder DSLs effektiv lernen und nachhaltig anwenden können. Dies steht in engem Zusammenhang mit Kernaussage 5, die eine einheitliche Terminologie und konsistente Teststruktur als Erfolgsfaktor hervorhebt. ExpertIn 5 sieht in Dokumentationen, Wikis und Hilfetexten eine wichtige Ergänzung, um den Einstieg zu erleichtern.

Die direkten Vorteile für Testabdeckung und Qualität (Kernaussage 8) wurden von 40 % der Expert:innen explizit hervorgehoben. Vor allem ExpertIn 2 und ExpertIn 5 betonen, dass dadurch mehr Use-Cases abgedeckt werden, an die ein/e reiner Testengineer:in gar nicht denken würde. In der Praxis bedeutet dies oftmals eine Verbesserung der Produktqualität, weil durch die interdisziplinäre Zusammenarbeit im Team vielfältige Perspektiven einfließen. Diese Diversität deckt kritische Edge-Cases, reale Nutzungsszenarien und potenzielle Fehlerquellen auf, die isoliertes Testen leicht übersehen könnte.

Abschließend weisen 40 % der Befragten (ExpertIn 3 und ExpertIn 5) auf Risiken (Kernaussage 9) durch falsche Annahmen oder eine unklare Rollenverteilung hin. „Man braucht meist noch einen Tester, der methodisch draufschaut und einen Entwickler für die technischen Details“, bemerkt ExpertIn 5. Es besteht die Gefahr, dass Fachexpert:innen ihre erstellten Testfälle als umfassend interpretieren, obwohl noch weitere Testaspekte fehlen. Diese Aussagen deuten darauf hin, dass das gemeinsame Testdesign gute Abstimmungsprozesse und klare Verantwortlichkeiten erfordert.

Insgesamt bestätigt die Auswertung, dass das Konzept von den Befragten als vielversprechender Ansatz gesehen wird, um Qualitätsaspekte früher in den Entwicklungsprozess zu integrieren. Die direkte Einbindung der Domänenexpert:innen und die Reduktion technischer Einstiegshürden werden als wesentliche Stärken genannt. Die Tabelle macht aber auch deutlich, dass organisatorische Maßnahmen wie Terminologie-Konzepte, Schulungen und ein klares Rollenverständnis von den Befragten als unabdingbar eingestuft werden, um langfristig von den Vorteilen zu profitieren. Gerade in komplexen Projekten steigt der Nutzen durch eine zielgerichtete Fach-Integration, gleichzeitig jedoch auch der Bedarf an strukturierter Koordination.

7.4 Zusammenfassung der Erkenntnisse

Aus den quantitativen und qualitativen Analysen ergibt sich eine insgesamt positive Grundhaltung gegenüber dem vorgestellten Konzept, das Fachexpert:innen in den Entwicklungs- und Testprozess einbindet. Die Mehrzahl der Befragten (80 %) sieht deutliche Vorteile in einer möglichst frühen Spezifikation von Abnahmekriterien, der Integration in CI/CD-Pipelines sowie in der direkten Beteiligung von Domänenexpert:innen.

Besonders in komplexen Projekten oder solchen mit hohem Fachanteil wird der Zugewinn an Qualität und Testabdeckung als entscheidend eingeschätzt. Diese Einschätzungen stehen in Einklang mit der Literatur, die betont, dass eine frühzeitige Einbindung von Domänenwissen entscheidend zur Verbesserung der Testabdeckung und Qualität beitragen kann, insbesondere in stark regulierten und technischen Umfeldern (siehe Kapitel 4.2).

Die qualitative Analyse bestätigt, dass die niedrige Einstiegshürde und die Gestaltung der Plattform entscheidende Faktoren für die Akzeptanz sind. Stakeholder ohne Programmierkenntnisse können Testfälle unmittelbar definieren und automatisieren, ohne Code schreiben zu müssen. Dies verkürzt die Abstimmung mit dem Entwicklungsteam und deckt potenzielle Missverständnisse früher auf. Studien über No-Code- und Low-Code-Plattformen bekräftigen diesen Punkt, indem sie hervorheben, dass intuitive Drag-and-Drop-Oberflächen die Nutzung durch Nicht-Programmierer:innen deutlich erleichtern und den Entwicklungsprozess beschleunigen (siehe Kapitel 3.4).

In Kombination mit einer CI-Integration kann der Ansatz einen kontinuierlichen Feedback-Zyklus ermöglichen, der Fehlentwicklungen reduziert und Wartungsaufwände minimiert. Die Literatur zu CI/CD-Prozessen hebt hervor, dass die Anbindung an Pipelines nicht nur die Automatisierung vorantreibt, sondern auch den gesamten Entwicklungsprozess agiler gestaltet und die Produktqualität verbessert (siehe Kapitel 2.9).

Die qualitative Analyse verweist jedoch auch auf konkrete Rahmenbedingungen, ohne die der Nutzen aus Sicht der Befragten deutlich sinken könnte. Besonders hervorgehoben werden:

- Die Notwendigkeit eines Konzepts, um Duplikate und Unklarheiten in den Testschritten zu vermeiden. Dies wird auch in Studien zu modellgetriebenen Ansätzen als zentraler Erfolgsfaktor benannt (siehe Kapitel 3.1).
- Das Erfordernis klarer Rollenverteilungen und Freigabeprozesse, um sicherzustellen, dass fachliche und technische Seiten effizient zusammenarbeiten. Rokis & Kirikova [110] betonen, dass Governance-Strukturen und standardisierte Prozesse essenziell sind, um die Effizienz von Low-Code/No-Code-Plattformen sicherzustellen.
- Die Bedeutung von Schulungen oder Workshops, um Fachexpert:innen das nötige Handwerkszeug für die Tool- oder DSL-Nutzung zu vermitteln. Die Literatur hebt hervor, dass ohne gezielte Schulungen die Potenziale solcher Plattformen nur begrenzt ausgeschöpft werden können (siehe Kapitel 3.5).

Mehrere Expert:innen wiesen darauf hin, dass sich nicht alle Szenarien rein „klickbasiert“ abbilden lassen. Besonders anspruchsvolle Use-Cases könnten nach wie vor ergänzendem Code oder tiefgehendem technischen Know-how bedürfen. Die Literatur zu modellbasierten Ansätzen weist ähnlich darauf hin, dass solche Konzepte in hardwarenahen oder komplexen Szenarien ohne Erweiterungsoptionen schnell an ihre Grenzen stoßen (siehe Kapitel 4.3).

Nach Auswertung der aktuellen Studien- und Literaturlage (vgl. Kapitel 3.4 & 4) lässt sich festhalten, dass viele bestehende Arbeiten zu No-/Low-Code-Ansätzen überwiegend auf softwarezentrierte Use-Cases ausgerichtet sind und hardwarebezogene Anforderungen entweder ausklammern oder nur sehr begrenzt behandeln. Zudem fokussieren sich zahlreiche Veröffentlichungen vornehmlich auf die technische Umsetzung (z. B. Erstellung einer No-/Low-Code-Plattform) und widmen den organisatorischen Rahmenbedingungen (z. B. Rollenmodelle, Terminologie-Standards, Versionierung von Testbausteinen) lediglich am Rande.

Die vorliegende Arbeit unterscheidet sich insbesondere durch die gezielte Integration des Hardware-Kontextes, den umfassenden Blick auf Governance und Organisation sowie die systematische Evaluation mit Expert:innen aus unterschiedlichen Fachgebieten. Während sich viele bestehende Studien vorrangig auf softwarezentrierte Testszenarien konzentrieren, werden hier bewusst Hardware-Schnittstellen und deren Einbindung in CI/CD-Pipelines berücksichtigt (Kapitel 6). Dieser Aspekt wird in der bisherigen Literatur nur selten ausführlich behandelt, da hardwarebezogene Testprozesse oft mit höheren Komplexitäten und eingeschränkter Reproduzierbarkeit verbunden sind [6].

Darüber hinaus liegt ein zentraler Fokus auf den organisatorischen Rahmenbedingungen, die für eine erfolgreiche und nachhaltige Nutzung von No-Code-Ansätzen erforderlich sind. Neben der rein technischen Implementierung werden wesentliche Erfolgsfaktoren wie eine geregelte Rollenverteilung, klare Review-Prozesse sowie ein konsistentes Terminologie- und Steps-Konzept analysiert (Kapitel 5). Die Interviews mit Fachleuten (Kapitel 7) verdeutlichen, dass diese Strukturen entscheidend sind, um eine langfristige Wartbarkeit und Qualitätssicherung der erstellten Testfälle zu gewährleisten.

Ein weiterer wesentlicher Unterschied zu bisherigen Arbeiten besteht in der methodischen Herangehensweise. Während viele Studien sich auf konzeptionelle Prototypen oder Einzelfallstudien beschränken, wird hier ein Ansatz verfolgt, der sowohl quantitative als auch qualitative Analysen kombiniert. Die Einbindung von Expert:innen aus den Bereichen Testmanagement, Softwareentwicklung, IT-Architektur und Projektleitung ermöglicht es, das Konzept aus verschiedenen Perspektiven zu bewerten und praxisnahe Erkenntnisse über die Umsetzbarkeit und Akzeptanz in realen Projekten zu gewinnen [126]. Dadurch liefert diese Arbeit nicht nur eine technische Machbarkeitsstudie, sondern betrachtet auch die organisatorischen und methodischen Anforderungen, die für eine erfolgreiche Einführung und Skalierung einer No-Code-Testautomatisierung notwendig sind.

Dadurch zeigt die Arbeit praxisnah, wie ein No-Code-Konzept im Zusammenspiel mit CI/CD- und Hardware-Komponenten aussehen kann und welche organisatorischen Voraussetzungen (siehe Kapitel 7.4) dafür nötig sind. Dieser integrierte Ansatz geht über die bestehenden Studien hinaus, die sich überwiegend auf reine Softwareprojekte oder auf punktuelle Low-Code-Implementierungen konzentrieren, ohne dabei eine ganzheitliche Einbettung in agile Entwicklungsprozesse und komplexe Domänen vorzunehmen. Demnach schafft die Arbeit eine erweiterte Basis für zukünftige Forschungs- und Pilotprojekte, indem sie konkrete Empfehlungen für die Skalierung und langfristige Wartung einer No-Code-Testautomatisierung in realen Hardware-Software-Projekten aufzeigt.

8 Diskussion

Dieses Kapitel widmet sich der kritischen Reflexion und Interpretation der in den Expert:inneninterviews (Kapitel 7) gewonnenen Erkenntnisse im Hinblick auf die Forschungsfragen F1 und F2 (vgl. Kapitel 1.2). Dabei werden sowohl Potenziale als auch Herausforderungen beleuchtet, die sich aus den Ergebnissen für den modellbasierten Testansatz ergeben. Im Anschluss wird die zentrale Hypothese dieser Arbeit überprüft und in den Kontext ihrer Limitationen gestellt. Schließlich folgt ein Ausblick, der Handlungsempfehlungen für die praktische Umsetzung ableitet, bevor das Kapitel mit einer zusammenfassenden Schlussfolgerung schließt.

8.1 Einordnung der Ergebnisse zu F1

Welche zentralen Anforderungen und Kriterien sollten modellbasierte Ansätze erfüllen, um die effiziente Erstellung und Wartung von automatisierten Tests für unterschiedliche Testtypen durch Fachexpert:innen ohne spezifische Programmierkenntnisse zu ermöglichen? (vgl. Kapitel 1.2)

Die befragten Expert:innen betonen, dass ein modellbasierter Ansatz vor allem dann erfolgreich ist, wenn er nicht lediglich auf einer benutzerfreundlichen Oberfläche für Fachexpert:innen beruht, sondern darüber hinaus bestimmte organisatorische und technische Kriterien erfüllt (vgl. Kapitel 7). Besonders hervorgehoben wird die frühzeitige Einbindung des Fachbereichs in den Entwicklungs- und Testprozess, da sich so Anforderungen schon in der Anforderungs- oder Designphase präzise formulieren lassen und spätere Korrekturschleifen vermieden werden können. Gleichzeitig erfordert die effiziente Erstellung und Wartung automatisierter Tests eine klare Rollen- und Prozessdefinition, damit Testschritte nicht mehrfach oder in widersprüchlicher Form angelegt werden und die Qualität insgesamt gewahrt bleibt. Um diese Konsistenz sicherzustellen, verweisen die Befragten immer wieder auf die Notwendigkeit eines strukturierten Freigabeprozesses und eines transparenten Rechtemanagements, insbesondere wenn mehrere Fachexpert:innen parallel an Testfällen oder Modellkomponenten arbeiten.

Ein weiterer zentraler Aspekt ist die konsequente Nachvollziehbarkeit aller Änderungen an Testschritten, Metamodellen oder Testbibliotheken. Durch eine lückenlose Versionierung dieser Artefakte wird es möglich, jederzeit zu sehen, welche Anpassungen vorgenommen wurden. Dies erleichtert die Durchführung von Regressionstests und ermöglicht es, bei Bedarf effizient auf frühere Projektstände zurückzugreifen. Expert:innen betonen, wie wichtig die nahtlose Integration solcher Ansätze in bestehende DevOps-Prozesse oder CI/CD-Pipelines ist. Kontinuierliche Tests und schnelles Feedback fördern nicht nur die Akzeptanz des Ansatzes im Team, sondern helfen auch, potenzielle Fehler frühzeitig zu erkennen. Darüber hinaus ist eine No-Code-Plattform erforderlich, die

keine tiefgreifenden Programmierkenntnisse voraussetzt, jedoch die Möglichkeit bietet, erweiterbare Funktionen für komplexere Szenarien zu verwenden. Insbesondere bei anspruchsvollen Tests mit Schleifen, Verzweigungen oder mehreren Schnittstellen sind reine Klickabläufe oft nicht ausreichend, um alle Anforderungen abzudecken. Die Befragten sehen einen großen Mehrwert in der Möglichkeit, komplexe oder speziellere Anwendungsfälle zu spezifizieren, an die ein rein technisch orientiertes Team unter Umständen nicht denken würde. Allerdings stellt sich diese höhere Testabdeckung nur dann ein, wenn Fachexpert:innen nicht nur anfangs, sondern kontinuierlich mitwirken und die Tests über den gesamten Lebenszyklus pflegen. In diesem Zusammenhang wird wiederholt darauf hingewiesen, dass Schulungen oder begleitende Workshops unverzichtbar sind, damit die Beteiligten die Grundprinzipien des modellbasierten Vorgehens kennen und konsequent anwenden können. Gerade in größeren Projekten sind Hilfesysteme wie Wikis oder In-Tool-Tipps hilfreich, um die Anwendung der Modellierungssprache oder des Testeditors zu standardisieren und Fehlerquellen zu minimieren.

Tabelle 5 fasst die zentralen Anforderungen aus den Interviews zusammen und hebt die Bedeutung eines klaren Governance-Konzepts sowie einer effektiven organisatorischen Einbettung hervor. Ein Governance-Konzept umfasst Richtlinien, Prozesse und Verantwortlichkeiten, die sicherstellen, dass Projekte zielgerichtet, effizient und risikobewusst umgesetzt werden [131]. Dabei wird insbesondere betont, wie wichtig eine enge Zusammenarbeit zwischen Fach- und IT-Abteilungen ist, um Synergien zu schaffen. Dies erfordert ein gemeinsames Verständnis der Anforderungen und eine frühzeitige Kommunikation potenzieller Risiken, um die Grundlage für eine erfolgreiche Projektdurchführung zu legen.

Zusammenfassend lassen die Interviewergebnisse erkennen, dass für die effiziente Erstellung und Wartung von automatisierten Tests ohne tiefgehende Programmierkenntnisse mehrere Voraussetzungen erfüllt sein müssen. Neben einer leicht verständlichen Benutzeroberfläche braucht es vor allem ein durchdachtes organisatorisches Rahmenwerk, das klare Rollen, Prozesse und Freigabemechanismen definiert, eine robuste technologische Basis mit Versionierung und CI-Anbindung bereitstellt und zugleich genügend Flexibilität für komplexere Testszenarien gewährleistet. Darüber hinaus trägt eine gezielte Einbindung der Fachseite zu einem früheren Aufdecken möglicher Fehlerquellen bei und erhöht die Testabdeckung in fachlich relevanten Bereichen. Gelingt es, all diese Anforderungen zu vereinen, erhoffen die Befragten sich eine deutliche Steigerung der Effizienz und Qualität in der Testautomatisierung bei Projekten mit Software- und Hardwarekomponenten.

Anforderung	**Konkretisierung**
Frühzeitige Einbindung der Fachexpert:innen	Einbeziehung bereits in der Anforderungs- oder Designphase, um Use-Cases und Abnahmekriterien frühzeitig zu definieren und Fehlerquellen zu minimieren
Klare Prozesse, Rollenverteilung und Rechtemanagement	Strukturierter Review-Prozess zur Freigabe neuer oder geänderter Testschritte, eindeutige Verantwortlichkeiten für Fach- und Technikseite, Vermeidung von Duplikaten und widersprüchlichen Tests

Versionierung und Freigabeprozess	Einsatz von Tools, die verschiedene Versionen der Testfälle verwalten, automatisierte Regressionstests ermöglichen und Änderungen lückenlos nachverfolgbar machen
Integration in DevOps- und CI/CD-Umgebungen	Anbindung an Continuous-Integration-Pipelines, Issue-Tracking-Systeme und Build-Prozesse, um kontinuierliche Testläufe und schnelles Feedback zu gewährleisten
Benutzerfreundliches Modellierungs-Interface mit Flexibilität	Klickbasierte Oberfläche oder DSL für einfache Szenarien, gleichzeitig Erweiterungsfunktionen für komplexe Abläufe wie Verzweigungen, Schleifen und Schnittstellenaufrufe
Schulungen und Onboarding	Workshops, Training und begleitende Dokumentationen (z. B. Wiki, Tooltipps) für Fachexpert:innen, damit das modellbasierte Vorgehen langfristig konsistent angewendet und gepflegt werden kann
Organisatorische Einbettung und Governance	Festlegung von Terminologie, standardisierten Testbausteinen und einem Qualitätskontrollmechanismus, um Konsistenz und Wartbarkeit des Testbestands zu sichern
Synergie zwischen Fach- und IT-Seite	Gemeinsames Verständnis der Anforderungen, frühzeitige Kommunikation über Testdesign und potenzielle Risiken, gemeinsame Verantwortung für die Pflege und Weiterentwicklung des Testmodells

Tabelle 5 Liste der Anforderungen zur Beantwortung von F1

8.2 Einordnung der Ergebnisse zu F2

Wie erleichtert ein modellbasierter Ansatz die Erstellung und Wartung von automatisierten Tests und welche Unterschiede, Limitierungen und Potenziale ergeben sich im Vergleich zu herkömmlichen Programmieransätzen? (vgl. Kapitel 1.2)

Die im Rahmen dieser Arbeit geführten Interviews (Kapitel 7) zeigen, dass ein modellbasierter Ansatz vor allem dadurch entlastet, dass Fachexpert:innen in einer klick- oder modellbasierten Umgebung frühzeitig an der Testfall-Definition mitwirken können. Da technisches Code-Wissen nur noch in begrenztem Umfang erforderlich ist, entsteht eine engere Verzahnung zwischen Fach- und IT-Seite, weil Abnahmekriterien und Domänenwissen schon in der Planungs- oder Designphase in konkrete Tests überführt werden. Rund 80 % der befragten Expert:Innen bewerten diesen Zugang positiv und sehen darin einen wichtigen Faktor für eine bessere Testabdeckung und eine frühe Fehlererkennung. Anders als bei rein programmatischen Ansätzen, bei denen Entwickler:innen

allein für die Codierung und Interpretation von Tests verantwortlich sind, wird das fachliche Know-how hier direkt im Modell abgebildet. Dadurch kann die Kommunikation zwischen den Beteiligten beschleunigt werden, da Tests nicht erst nachträglich aus Textdokumenten oder persönlichen Gesprächen abgeleitet werden müssen.

Die Wartung automatisierter Tests wird nach Ansicht vieler Teilnehmender ebenfalls erleichtert, weil eine Änderung an Geschäftsregeln oder Use Cases zentral im Modell stattfinden kann. Insbesondere in komplexen Projekten, wo ein durchgängiges Terminologie-Konzept essenziell ist, lassen sich redundante Testschritte vermeiden, wenn das Modell klar strukturiert und für alle beteiligten Rollen einsehbar ist. In herkömmlichen Code- oder Skript-Frameworks wären diese Änderungen häufig auf mehrere Dateien verteilt und für Fachexpert:innen nur mit Unterstützung von Entwickler:innen zu bewältigen. Der modellbasierte Ansatz kann zudem leichter in bestehende CI/CD-Pipelines integriert werden und ermöglicht so ein fortlaufendes Feedback im Entwicklungsprozess. Dies wird von 80 % der Befragten als wichtig bis sehr wichtig bewertet, weil sowohl Fachleute als auch Entwickler:innen unmittelbar sehen, ob das System auf neu entstandene oder geänderte Testfälle reagiert.

Trotz dieser Vorteile nennen einige der befragten Expert:innen auch Grenzen, die sich etwa in hochkomplexen oder hardware-nahen Testfällen zeigen. Reine Klick- oder No-Code-Lösungen stoßen hier ohne Erweiterungsmöglichkeiten oder einen klaren Governance-Rahmen [131] schnell an ihre Grenzen. Auch ist ein gewisser initialer Mehraufwand nicht zu unterschätzen, da die Modellierung, Pflege und Versionierung von Testbausteinen koordiniert werden müssen, ehe man wirklich von einer zentralen Wartung profitieren kann. In einem rein codebasierten Setting besteht zwar oft mehr Flexibilität für spezielle Logiken, doch dafür haben viele Fachexpert:innen keine direkte Möglichkeit, Tests eigenständig zu entwerfen oder zu aktualisieren. Die Befragten weisen zudem darauf hin, dass selbst bei hoher Akzeptanz ein strukturierter Review-Prozess unverzichtbar bleibt, um Missverständnisse in der Umsetzung zu vermeiden und rollenbasierte Verantwortlichkeiten klar festzulegen.

Der modellbasierte Ansatz erleichtert die Testautomatisierung insbesondere durch seine niedrigere technische Einstiegshürde, die zentrale Pflege aller relevanten Abläufe und die bessere Sichtbarkeit fachlicher Anforderungen. Im Vergleich zu herkömmlichen Programmieransätzen wird somit eine stärkere Beteiligung des Fachbereichs ermöglicht, was sich positiv auf die Testabdeckung und die Reaktionsgeschwindigkeit im Entwicklungsprozess auswirken kann. Werden jedoch keine klaren Regeln für Terminologie, Rollenverteilung und Qualitätssicherung definiert, können sich die Vorteile schnell relativieren. Gerade in Szenarien mit hochgradigen Abhängigkeiten oder umfangreicher Hardwareintegration ist es laut den Befragten entscheidend, dass die Modellierungsumgebung flexible Erweiterungs-Optionen bietet und von einem methodischen Vorgehen begleitet wird, um die Stärken gegenüber rein codebasierten Testverfahren konsequent auszuspielen.

8.3 Überprüfung der Hypothese

Die Hypothese dieser Arbeit lautet, dass *„modellbasierte Ansätze für die Erstellung und Wartung automatisierter Tests es Fachexpert:innen ohne Programmierkennt-*

***nisse ermöglichen, aktiv am Entwicklungsprozess teilzunehmen, was die Testabdeckung erhöht und die Lücke zwischen fachlicher und technischer Expertise überbrückt*“. (vgl. Kapitel 1.2)**

Die Auswertung der Expert:inneninterviews zu F1 und F2 legt insgesamt nahe, dass die aufgestellte Hypothese in ihren wesentlichen Punkten bestätigt werden kann. F1 zeigt, welche Anforderungen und Kriterien ein modellbasierter Ansatz erfüllen sollte, damit Fachexpert:innen ohne spezifische Programmierkenntnisse aktiv in die Erstellung und Wartung von automatisierten Tests eingebunden werden können. Insbesondere die frühzeitige Einbindung des Fachbereichs, klare Prozesse und Rollenverteilungen sowie eine robuste technische Infrastruktur (etwa für Versionierung, CI-Integration und erweiterbare Modellierungs-Interfaces) gelten als essenzielle Voraussetzungen. F2 ergänzt dieses Bild, indem es verdeutlicht, wie ein modellbasierter Ansatz konkret zu einer engeren Verzahnung von Fach- und IT-Seite und damit zu einer verbesserten Testabdeckung führt. Die befragten Expert:innen betonen, dass Tests auf diese Weise nicht erst nachträglich codiert, sondern direkt aus den fachlichen Anforderungen heraus modelliert werden können. Dadurch werden mögliche Fehlinterpretationen reduziert und ein kontinuierliches Feedback im Entwicklungsprozess ermöglicht.

Die in F2 genannten Vorteile – zum Beispiel die zentrale Pflege fachlicher Abläufe im Modell oder die vereinfachte Wartung – stützen die zweite Komponente der Hypothese, wonach das Einfließen fachlichen Know-hows die Testabdeckung erhöht und die Lücke zwischen technischer und fachlicher Expertise verringert. Die Befragten weisen jedoch gleichzeitig darauf hin, dass ein reines Klick- oder No-Code-Verfahren in hochkomplexen Szenarien an seine Grenzen stoßen kann und dass ein methodisches Vorgehen mit geeigneten Governance-Strukturen unabdingbar ist. Die Hypothese, wonach ein modellbasierter Ansatz ohne tiefgehende Programmierkenntnisse realisierbar ist, wird somit bejaht, allerdings stets unter der Bedingung, dass ausreichend organisatorische Leitlinien existieren und die benötigte Tool-Unterstützung bereitsteht.

Die Interviews bestätigen, dass modellbasierte Testansätze es Fachexpert:innen erleichtern, im Entwicklungsprozess mitzuwirken, sobald ein durchdachtes Zusammenspiel technischer, organisatorischer und personeller Aspekte etabliert ist. Die Hypothese wird damit weitgehend unterstützt, erfordert jedoch die Einschränkung, dass weder die Kommunikation noch die Qualitätssicherung automatisch gewährleistet sind. Vielmehr sind klare Verantwortlichkeiten, ein strukturiertes Review- und Freigabekonzept sowie kontinuierliche Schulungsangebote erforderlich, um das Potenzial des modellbasierten Vorgehens voll auszuschöpfen und einen spürbaren Mehrwert gegenüber rein codebasierten Testverfahren zu erzielen.

8.4 Limitationen und kritische Reflexion

Die Ergebnisse der vorliegenden Arbeit sind insgesamt positiv und zeigen das Potenzial des entwickelten Konzeptes auf. Dennoch ist es essenziell, die Limitationen der Untersuchung und die damit verbundenen Implikationen kritisch zu reflektieren, um die Aussagekraft der Studie realistisch einzuordnen.

Eine der zentralen Einschränkungen dieser Studie liegt in der geringen Anzahl der interviewten Expert:innen (n = 5). Diese kleine Stichprobe reduziert die Generalisierbarkeit der Ergebnisse, da die Einschätzungen und Meinungen nur einen Ausschnitt der relevanten Fachwelt repräsentieren. Eine größere und diversere Gruppe von Teilnehmenden könnte wertvolle, differenziertere Perspektiven liefern. Insbesondere wären Meinungen von Stakeholdern mit unterschiedlichem Erfahrungshintergrund oder solchen aus anderen Domänen relevant, um ein umfassenderes Bild der Anwendbarkeit des Konzepts zu erhalten.

Ein weiterer Punkt ist die potenzielle Verzerrung durch die Voreinstellung der Befragten. Die Mehrheit der Teilnehmenden zeigte sich bereits vor Beginn der Untersuchung aufgeschlossen gegenüber Automatisierungskonzepten. Diese positive Grundhaltung könnte die Ergebnisse zugunsten des entwickelten Modellierungsansatzes beeinflusst haben, da kritische Perspektiven unterrepräsentiert blieben. Für zukünftige Studien wäre es sinnvoll, gezielt Personen mit skeptischen oder kritischen Standpunkten einzubeziehen, um ein ausgewogenes Meinungsbild zu erhalten.

Sowohl in den quantitativen als auch qualitativen Methoden sind Verzerrungen nie vollständig auszuschließen. Eine mögliche Quelle solcher Verzerrungen war die Vorführung des Prototyps durch den Autor selbst. Diese Konstellation könnte unbeabsichtigt die Antworten der Befragten beeinflusst haben, beispielsweise durch subtile Hinweise oder durch die Erwartung, dass der Autor ein positives Feedback bevorzugt. Hier wäre es in zukünftigen Untersuchungen hilfreich, eine neutrale Person für die Demonstration einzusetzen.

Der modellbasierte Ansatz wurde lediglich über einen kurzen Zeitraum evaluiert. Die langfristige Wirksamkeit und Nachhaltigkeit des Konzeptes in unterschiedlichen Kontexten bleiben somit offen. Besonders in großen Projekten oder über mehrere Produktinkremente hinweg könnten unerwartete Herausforderungen auftreten. Langzeitstudien wären notwendig, um die Skalierbarkeit und Robustheit des Ansatzes zu überprüfen und mögliche Schwächen in der Praxis aufzudecken.

Ein spezifischer Aspekt, der in die kritische Reflexion einbezogen werden muss, sind die Herausforderungen und Risiken im Umgang mit hardwaregestützten Testumgebungen. Die Integration dedizierter Hardware in CI/CD-Pipelines bietet zwar eine realitätsnahe Testumgebung, bringt jedoch erhebliche Abhängigkeiten mit sich. Die begrenzte Anzahl physischer Ressourcen und deren Wartungsanforderungen stellen potenzielle Engpässe dar. Fällt beispielsweise eine zentrale Hardwarekomponente aus, können Testprozesse unterbrochen und Entwicklungszeiten verzögert werden. Zur Minimierung dieser Risiken wäre eine sorgfältige Planung der Hardwareinfrastruktur notwendig. Dies schließt die Skalierung der Kapazitäten, den Aufbau redundanter Systeme sowie effek-

tive Backup-Strategien ein. Darüber hinaus könnte ein umfassendes Monitoring frühzeitig potenzielle Probleme erkennen und vorbeugende Maßnahmen erleichtern. Ein solches Vorgehen würde nicht nur die Stabilität des Testbetriebs erhöhen, sondern auch das Risiko minimieren, dass einzelne Fehler den gesamten Entwicklungsprozess beeinträchtigen.

Zusammenfassend verdeutlichen die aufgezeigten Limitationen, dass die Ergebnisse der Studie mit Vorsicht interpretiert werden sollten. Während die vorgestellten Ansätze vielversprechend sind, besteht weiterer Forschungsbedarf, insbesondere hinsichtlich der Generalisierbarkeit, der Berücksichtigung kritischer Perspektiven und der langfristigen Praxistauglichkeit des Konzepts. Zusätzlich sind spezifische Herausforderungen in hardwaregestützten Umgebungen bei der Planung und Umsetzung solcher Ansätze zu beachten. Eine erweiterte Evaluation könnte dazu beitragen, diese Schwächen zu adressieren und die Robustheit sowie Anwendbarkeit des Modellierungsansatzes weiter zu verbessern.

9 Zusammenfassung und Ausblick

Die Zusammenfassung und der Ausblick dieses Kapitels setzen sich mit den zentralen Ergebnissen und den weiterführenden Implikationen der Arbeit auseinander. Es wird deutlich, wie das entwickelte Konzept und die prototypische Umsetzung dazu beitragen können, die Qualitätssicherung in Projekten mit Software- und Hardwarekomponenten zu transformieren. Gleichzeitig werden die identifizierten Herausforderungen beleuchtet, die als Grundlage für zukünftige Forschungs- und Entwicklungsansätze dienen können.

9.1 Zusammenfassung

Die Diplomarbeit widmet sich der Integration von Fachexpert:innen ohne Programmierkenntnisse in die automatisierte Qualitätssicherung. Sie untersucht, wie durch die Entwicklung einer No-Code-Plattform die Erstellung und Wartung von Tests in Projekten mit Software- und Hardwarekomponenten erleichtert werden kann. Die Arbeit liefert dabei nicht nur theoretische Einsichten, sondern bietet mit der prototypischen Implementierung eine praxisorientierte Grundlage, die den Nutzen und die Herausforderungen dieses Ansatzes beleuchtet.

Ausgangspunkt ist die Erkenntnis, dass traditionelle Methoden der Qualitätssicherung in Projekten mit Hardwareintegration oft nicht ausreichend flexibel und effizient sind, um den steigenden Anforderungen gerecht zu werden. Komplexität, Repetitivität und die Notwendigkeit, interdisziplinäres Fachwissen zu berücksichtigen, stellen dabei wesentliche Hindernisse dar. Insbesondere in Szenarien, in denen Hardwarekomponenten getestet werden müssen, ist eine präzise und wiederholbare Automatisierung von großem Wert (Kapitel 1.1).

Diese Arbeit setzt hier an, indem sie ein Konzept vorstellt, das es Fachexpert:innen ermöglicht, Testfälle in einer Drag-and-Drop-Umgebung zu erstellen, ohne technische Vorkenntnisse zu benötigen. Dies geschieht auf Basis einer prototypischen Plattform, die nahtlos in bestehende CI/CD-Pipelines integriert werden kann und dabei nicht nur die Automatisierung von Testläufen ermöglicht, sondern auch eine intuitive Fehleranalyse und eine optimierte Zusammenarbeit zwischen Fach- und Entwicklungsteams unterstützt (Kapitel 6).

Ein wichtiger Punkt der Arbeit liegt auf der Ermittlung der Anforderungen an ein solches System. Mithilfe von systematischen Literaturrecherchen und Expert:inneninterviews wurden zentrale Kriterien identifiziert, die eine derartige Plattform erfüllen muss (Kapitel 5). Dazu zählen unter anderem die flexible Anpassbarkeit an verschiedene Testszenarien, die Erweiterbarkeit der Plattform durch neue Befehle und die Möglichkeit, auch komplexe Abläufe einfach zu modellieren (Kapitel 5.3).

Der entwickelte Prototyp (Kapitel 6) illustriert die praktische Umsetzung des Konzepts: Mit einer klar strukturierten Benutzeroberfläche können Nutzer:innen Testfälle aus vordefinierten Bausteinen zusammenstellen, ausführen und analysieren. Die Plattform bietet darüber hinaus ein integriertes Wiki, das die Zusammenarbeit und Wissensweitergabe zwischen den Beteiligten fördert. Die Ergebnisse werden in einem Dashboard übersichtlich präsentiert, wobei ein automatisches Log-Analyse-Tool die Testergebnisse in eine für die Fachanwender:innen verständliche Sprache übersetzt.

Die Arbeit verdeutlicht, wie modellbasierte Ansätze zur Testautomatisierung eine Brücke zwischen technischer und fachlicher Expertise erschaffen können. Sie zeigt auf, dass diese Ansätze nicht nur eine effizientere Qualitätssicherung ermöglichen, sondern auch die Zusammenarbeit in interdisziplinären Teams nachhaltig verbessern. Die durchgeführten Expert:inneninterviews bestätigen, dass eine frühzeitige und enge Einbindung von Fachexpert:innen in den Entwicklungsprozess zentrale Vorteile mit sich bringt: Missverständnisse können frühzeitig aufgedeckt, Abnahmekriterien präzise definiert und spätere Korrekturen vermieden werden. Dies trägt sowohl zu einer höheren Testabdeckung als auch zu einer verbesserten Kommunikation zwischen Fach- und Entwicklungsteams bei. Insbesondere in frühen Projektphasen bietet der modellbasierte Ansatz einen klaren Mehrwert, da er Fachexpert:innen erlaubt, Anforderungen und Use-Cases aktiv mitzugestalten, ohne tiefgehende Programmierkenntnisse zu benötigen. Gleichzeitig unterstützt die Integration in Continuous-Integration-Prozesse eine kontinuierliche Rückkopplung, die die Qualität von Software- und Hardwareprojekten nachhaltig steigert. Damit zeigt die Arbeit, dass technologische Innovation nicht nur Prozesse optimieren, sondern auch die Zusammenarbeit zwischen verschiedenen Disziplinen fördern und den Weg für integrative, zukunftsweisende Methoden in der Softwareentwicklung ebnen kann (Kapitel 7).

Die Arbeit diskutiert auch kritisch die Limitationen des Ansatzes (Kapitel 8.4). Insbesondere die Herausforderungen bei der Wartung komplexer Testszenarien, der Integration dedizierter Hardware in CI/CD-Pipelines und der Notwendigkeit klarer Governance-Modelle werden thematisiert. Dennoch überwiegen die positiven Effekte, die sich insbesondere durch die Flexibilität und Erweiterbarkeit des Systems ergeben. Der Prototyp legt so den Grundstein für eine zukunftsweisende Herangehensweise, die in verschiedenen Branchen Anwendung finden kann. Es zeigt sich auch, dass ein reines Klickmodell allein nicht ausreicht, um einen nachhaltigen Effizienz- und Qualitätsgewinn zu realisieren. Vielmehr ist ein Zusammenspiel aus klaren Rollen- und Governance-Regelungen, technischer Flexibilität, umfassenden Versionierungskonzepten sowie einer gut geplanten Tool-Integration in bestehende DevOps- und CI/CD-Umgebungen erforderlich. Gelingt diese Verzahnung, so ist davon auszugehen, dass die Vorteile des modellbasierten Ansatzes – etwa eine verbesserte Nachvollziehbarkeit von Testfällen, mehr Transparenz für alle Beteiligten und eine höhere Testabdeckung in spezifischen Domänenszenarien – langfristig zum Tragen kommen (Kapitel 8).

Insgesamt lassen die Ergebnisse darauf schließen, dass der hier vorgestellte Ansatz auch für komplexe Projekte vielversprechend sein kann, sofern die organisatorischen und technischen Voraussetzungen konsequent umgesetzt werden. Gerade durch mögliche Erweiterungen oder verfeinerte Rollenmodelle ergeben sich zusätzliche Chancen, den

Ansatz weiter an die spezifischen Bedürfnisse verschiedener Branchen und Projektkontexte anzupassen. Damit kann diese Arbeit als solide Grundlage für weitere Forschungen und Pilotprojekte dienen, welche die modellbasierte Testautomatisierung in noch größerem Maßstab evaluieren und die langfristige Nachhaltigkeit dieses Ansatzes untersuchen.

9.2 Ausblick und Empfehlungen

Ein modellbasierter Ansatz für die Testautomatisierung birgt ein enormes Potenzial, um die Zusammenarbeit zwischen Fach- und Entwicklungsteams zu fördern und die langfristige Wartbarkeit von Tests nachhaltig zu verbessern. Damit dieses Potenzial voll ausgeschöpft werden kann, ist es entscheidend, gezielte Weiterentwicklungen und Forschungen voranzutreiben.

Ein wichtiger Ansatzpunkt ist die Evaluation des Konzepts in Langzeit- und Skalierungsstudien. Diese sollten sich über mehrere Projektphasen erstrecken und idealerweise in Domänen mit hoher fachlicher Komplexität sowie verteilten Systemen mit heterogenen Hardwarekomponenten stattfinden. Solche Studien könnten aufzeigen, wie belastbar und skalierbar der Ansatz ist, wenn die Komplexität von Projekten zunimmt oder regulatorische Vorgaben erfüllt werden müssen.

Gleichzeitig muss der Fokus auf die nahtlose Integration in bestehende Umgebungen gelegt werden. Unternehmen nutzen etablierte Workflows, die durch den modellbasierten Ansatz nicht ersetzt, sondern ergänzt werden sollten. Nur wenn der Ansatz sich harmonisch in diese bestehenden Prozesse einfügt, können Automatisierung und Dokumentation durchgängig gewährleistet werden. Dies ist essenziell, um Akzeptanzbarrieren im Team abzubauen und den Nutzen der Plattform voll zur Geltung zu bringen.

Neben technologischen Innovationen sind auch organisatorische Rahmenbedingungen von zentraler Bedeutung. Ein durchdachtes Rollenmodell sowie klare Governance-Prozesse sorgen dafür, dass Fachexpert:innen und Entwickler:innen effektiv zusammenarbeiten und Testfälle konsistent aufgebaut und gepflegt werden. Ohne diese Strukturen drohen Redundanzen, Wissenslücken und eine ineffiziente Nutzung der Plattform. Dies könnte den Mehrwert des Ansatzes erheblich schmälern.

Um die Akzeptanz und den eigenständigen Umgang mit der Plattform zu fördern, sollte ein interdisziplinäres Schulungskonzept entwickelt werden. Neben klassischen Workshops könnten Online-Kurse, Wikis und interaktive In-Tool-Hilfen Fach- und Entwicklungsteams gleichermaßen unterstützen. Diese Maßnahmen tragen dazu bei, das Potenzial der Plattform nicht nur punktuell, sondern langfristig auszuschöpfen.

Durch gezielte Forschung, konsequente Weiterentwicklung und eine enge Verzahnung mit bestehenden Prozessen lässt sich der modellbasierte Ansatz zu einem zentralen Bestandteil moderner Qualitätssicherungssysteme entwickeln. Diese Vision erfordert jedoch eine klare Strategie, um das Zusammenspiel von Technologie, Organisation und

Schulung optimal zu gestalten und so den größtmöglichen Nutzen für Unternehmen und ihre Projekte zu generieren.

10 Literaturverzeichnis

[1] P. Dhore, L. Wadhwa, P. Shinde, D. Chaudhri, und P. Vyas, „BRIEF REVIEW ON DIFFERENT MANUAL SOFTWARE TESTING APPROACHES & PROCEDURE“, *J. Pharm. Negat. Results*, S. 455–464, Jan. 2023, doi: 10.47750/pnr.2023.14.S02.56. Verfügbar unter: https://pnrjournal.com/index.php/home/article/view/6707. [Zugegriffen: 4. Juli 2023]

[2] D. M. Rafi, K. R. K. Moses, K. Petersen, und M. V. Mäntylä, „Benefits and limitations of automated software testing: Systematic literature review and practitioner survey“, in *2012 7th International Workshop on Automation of Software Test (AST)*, 2012, S. 36–42. Verfügbar unter: https://ieeexplore.ieee.org/document/6228988. [Zugegriffen: 28. Februar 2025]

[3] R. Ramler und K. Wolfmaier, „Economic perspectives in test automation: balancing automated and manual testing with opportunity cost“, in *Proceedings of the 2006 international workshop on Automation of software test*, 2006, S. 85–91. Verfügbar unter: https://dl.acm.org/doi/10.1145/1138929.1138946. [Zugegriffen: 28. Februar 2025]

[4] A. N. Ghazi, K. Petersen, und J. Börstler, „Heterogeneous systems testing techniques: An exploratory survey“, *Lect. Notes Bus. Inf. Process.*, Bd. 200, S. 67–85, 2015, doi: 10.1007/978-3-319-13251-8_5/TABLES/7. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-319-13251-8_5. [Zugegriffen: 28. Februar 2025]

[5] J. P. Dias, F. Couto, A. C. R. Paiva, und H. S. Ferreira, „A brief overview of existing tools for testing the internet-of-things“, in *2018 IEEE international conference on software testing, verification and validation workshops (ICSTW)*, 2018, S. 104–109. Verfügbar unter: https://www.researchgate.net/publication/326495937_A_Brief_Overview_of_Existing_Tools_for_Testing_the_Internet-of-Things. [Zugegriffen: 28. Februar 2025]

[6] R. Donini, S. Marrone, N. Mazzocca, A. Orazzo, D. Papa, und S. Venticinque, „Testing complex safety-critical systems in SOA context“, in *2008 International Conference on Complex, Intelligent and Software Intensive Systems*, 2008, S. 87–93. Verfügbar unter: https://ieeexplore.ieee.org/document/4606667. [Zugegriffen: 28. Februar 2025]

[7] E. H. Grosse, C. H. Glock, M. Y. Jaber, und W. P. Neumann, „Incorporating human factors in order picking planning models: framework and research opportunities“, *Int. J. Prod. Res.*, Bd. 53, Nr. 3, S. 695–717, Feb. 2015, doi: 10.1080/00207543.2014.919424. Verfügbar unter: http://www.tandfonline.com/doi/abs/10.1080/00207543.2014.919424. [Zugegriffen: 28. Februar 2024]

[8] P. Hofmann, C. Samp, und N. Urbach, „Robotic process automation“, *Electron. Mark.*, Bd. 30, Nr. 1, S. 99–106, März 2020, doi: 10.1007/s12525-019-00365-8. Verfügbar unter: http://link.springer.com/10.1007/s12525-019-00365-8. [Zugegriffen: 4. Juli 2023]

[9] R. Waszkowski, „Low-code platform for automating business processes in manufacturing“, *IFAC-Pap.*, Bd. 52, Nr. 10, S. 376–381, Jan. 2019, doi: 10.1016/j.ifacol.2019.10.060. Verfügbar unter: https://www.sciencedirect.com/science/article/pii/S2405896319309152. [Zugegriffen: 21. Juli 2024]

[10] J. Straub, „Automated testing and quality assurance of 3D printing/3D printed hardware: Assessment for quality assurance and cybersecurity purposes“, in *2016 IEEE AUTOTESTCON*, 2016, S. 1–5. doi: 10.1109/AUTEST.2016.7589613. Verfügbar unter: https://ieeexplore.ieee.org/document/7589613. [Zugegriffen: 28. Februar 2025]

[11] D. Wang, B. Barnwell, und M. B. Witt, „A Cross Platform Test Management System for the SUDAAN Statistical Software Package“, in *2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications*, 2009, S. 237–244. doi: 10.1109/SERA.2009.32. Verfügbar unter: https://ieeexplore.ieee.org/document/5381758. [Zugegriffen: 28. Februar 2025]

[12] R. Dong, Z. Huang, I. I. Lam, Y. Chen, und X. Wang, „WebRobot: web robotic process automation using interactive programming-by-demonstration“, in *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2022, S. 152–167. Verfügbar unter: https://dl.acm.org/doi/10.1145/3519939.3523711. [Zugegriffen: 28. Februar 2025]

[13] T. Grechenig, Hrsg., *Softwaretechnik: mit Fallbeispielen aus realen Entwicklungsprojekten*. in it Informatik. München: Pearson Studium, 2010. Verfügbar unter: https://books.google.at/books?id=RcSvlD2-QQ0C&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false. [Zugegriffen: 28. Februar 2025]

[14] „ISO 25010“. Verfügbar unter: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Zugegriffen: 10. März 2024]

[15] 14:00-17:00, „ISO/IEC 25040:2011“, *ISO*. Verfügbar unter: https://www.iso.org/standard/35765.html. [Zugegriffen: 10. März 2024]

[16] B. Boehm und V. R. Basili, „Top 10 list [software development]“, *Computer*, Bd. 34, Nr. 1, S. 135–137, Jan. 2001, doi: 10.1109/2.962984. Verfügbar unter: http://ieeexplore.ieee.org/document/962984/. [Zugegriffen: 7. Januar 2024]

[17] S. McConnell, *Code complete*, Second edition. Redmond, Washington: Microsoft Press, 2004. Verfügbar unter: https://github.com/media-lib/prog_lib/blob/master/general/Steve%20McConnell%20-%20Code%20Complete%20(2nd%20edition).pdf. [Zugegriffen: 28. Februar 2025]

[18] *Practical Software Testing*. in Springer Professional Computing. New York: Springer-Verlag, 2003. doi: 10.1007/b97392. Verfügbar unter: http://link.springer.com/10.1007/b97392. [Zugegriffen: 6. Juli 2024]

[19] G. J. Myers, C. Sandler, und T. Badgett, *The art of software testing*, 3rd ed. Hoboken, N.J: John Wiley & Sons, 2012. Verfügbar unter: https://malenezi.github.io/malenezi/SE401/Books/114-the-art-of-software-testing-3-edition.pdf. [Zugegriffen: 28. Februar 2025]

[20] A. Kramer und B. Legeard, *Model-Based Testing Essentials: Guide to the ISTQB® Certified Model-Based Tester Foundation Level*, 1. Aufl. Wiley, 2016.

doi: 10.1002/9781119130161. Verfügbar unter: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119130161. [Zugegriffen: 10. Januar 2024]

[21] C. Pardo, F. J. Pino, F. García, und M. Piattini, „Harmonizing Quality Assurance Processes and Product Characteristics“, *Computer*, Bd. 44, Nr. 6, S. 94–96, Juni 2011, doi: 10.1109/MC.2011.178. Verfügbar unter: https://ieeexplore.ieee.org/document/5875942. [Zugegriffen: 13. September 2024]

[22] „Cost of Poor Software Quality in the U.S.: A 2022 Report“, *CISQ*. Verfügbar unter: https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/. [Zugegriffen: 29. August 2023]

[23] L. Xavier, J. E. Montandon, und M. T. Valente, „Comments or Issues: Where to Document Technical Debt“, *IEEE Softw.*, Bd. 39, Nr. 5, S. 84–91, Sep. 2022, doi: 10.1109/MS.2022.3170825. Verfügbar unter: https://ieeexplore.ieee.org/document/9763563/. [Zugegriffen: 5. Juli 2023]

[24] J. C. Rocha, V. Zapalowski, und I. Nunes, „Understanding Technical Debt at the Code Level from the Perspective of Software Developers“, in *Proceedings of the XXXI Brazilian Symposium on Software Engineering*, in SBES '17. New York, NY, USA: Association for Computing Machinery, Sep. 2017, S. 64–73. doi: 10.1145/3131151.3131164. Verfügbar unter: https://doi.org/10.1145/3131151.3131164. [Zugegriffen: 13. September 2024]

[25] Z. He, C. Liu, H. Yan, und H. He, „A Formal Definition of Software Testing Based on Fuzzy Measure“, in *2013 Fourth World Congress on Software Engineering*, Hong Kong, China: IEEE, Dez. 2013, S. 59–63. doi: 10.1109/WCSE.2013.13. Verfügbar unter: http://ieeexplore.ieee.org/document/6754263/. [Zugegriffen: 7. Januar 2024]

[26] H. Jin und F. Zeng, „Research on the definition and model of software testing quality“, in *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, Guiyang, China: IEEE, Juni 2011, S. 639–644. doi: 10.1109/ICRMS.2011.5979352. Verfügbar unter: http://ieeexplore.ieee.org/document/5979352/. [Zugegriffen: 7. Januar 2024]

[27] ISTQB, „ISTQB Syllabus PDF“, . *August*, 2023, Verfügbar unter: https://www.istqb.org/certifications/certified-tester-foundation-level. [Zugegriffen: 13. September 2024]

[28] C. Pi, „Development and Classification of Computer Software Testing Technology“, *J. Phys. Conf. Ser.*, Bd. 1650, Nr. 3, S. 032111, Okt. 2020, doi: 10.1088/1742-6596/1650/3/032111. Verfügbar unter: https://dx.doi.org/10.1088/1742-6596/1650/3/032111. [Zugegriffen: 13. September 2024]

[29] M. Tuteja und G. Dubey, „A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models“, 2012. Verfügbar unter: https://www.semanticscholar.org/paper/A-Research-Study-on-importance-of-Testing-and-in-(-Tuteja-Dubey/605c413a3bd7eb537338480113b33ea911f3987b. [Zugegriffen: 9. Januar 2024]

[30] M. Kassab, J. F. DeFranco, und P. A. Laplante, „Software Testing: The State of the Practice“, *IEEE Softw.*, Bd. 34, Nr. 5, S. 46–52, 2017, doi: 10.1109/MS.2017.3571582. Verfügbar unter: https://ieeexplore.ieee.org/document/8048665/. [Zugegriffen: 8. Januar 2024]

[31] M. Cohn, „The Forgotten Layer of the Test Automation Pyramid“, 17. Dezember 2009. Verfügbar unter: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid. [Zugegriffen: 10. März 2024]

[32] S. SugunaMallika, „EATOOS-Testing Tool for Unit Testing of Object Oriented Software“, *Int. J. Comput. Appl.*, Bd. 80, Nr. 4, S. 6–10, Okt. 2013, doi: 10.5120/13847-1679. Verfügbar unter: http://research.ijcaonline.org/volume80/number4/pxc3891679.pdf. [Zugegriffen: 7. Juli 2024]

[33] D. Hamlet, „Subdomain testing of units and systems with state“, in *Proceedings of the 2006 international symposium on Software testing and analysis*, Portland Maine USA: ACM, Juli 2006, S. 85–96. doi: 10.1145/1146238.1146249. Verfügbar unter: https://dl.acm.org/doi/10.1145/1146238.1146249. [Zugegriffen: 7. Juli 2024]

[34] Y. Sun *u. a.*, „Automatic integration testing through collaboration diagram and logic contracts“, *J. Phys. Conf. Ser.*, Bd. 1187, Nr. 4, S. 042043, Apr. 2019, doi: 10.1088/1742-6596/1187/4/042043. Verfügbar unter: https://iopscience.iop.org/article/10.1088/1742-6596/1187/4/042043. [Zugegriffen: 7. Juli 2024]

[35] T. Bennett und P. Wennberg, „Maintaining verification test consistency between executable specifications and embedded software in a virtual system integration laboratory environment“, in *28th Annual NASA Goddard Software Engineering Workshop, 2003. Proceedings.*, Greenbelt, Maryland, USA: IEEE, 2003, S. 221–228. doi: 10.1109/SEW.2003.1270747. Verfügbar unter: http://ieeexplore.ieee.org/document/1270747/. [Zugegriffen: 7. Juli 2024]

[36] F. Wotawa, L. Klampfl, und L. Jahaj, „A framework for the automation of testing computer vision systems“, in *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, Mai 2021, S. 121–124. doi: 10.1109/AST52587.2021.00023. Verfügbar unter: https://ieeexplore.ieee.org/document/9462996. [Zugegriffen: 7. Juli 2024]

[37] N. Alamsyah, Budiman, und Titan Parama, „Analysis of E-learning user Acceptance using the Technology Acceptance Model (TAM) and end-User Computing Satisfaction (EUCS)“, *Formosa J. Appl. Sci.*, Bd. 2, Nr. 8, S. 1873–1892, Aug. 2023, doi: 10.55927/fjas.v2i8.5405. Verfügbar unter: https://journal.formosapublisher.org/index.php/fjas/article/view/5405. [Zugegriffen: 7. Juli 2024]

[38] 14:00-17:00, „ISO/IEC/IEEE 24765:2010“, *ISO*. Verfügbar unter: https://www.iso.org/standard/50518.html. [Zugegriffen: 30. August 2023]

[39] S. Joshi und I. Kumari, „Analyses of Software Testing Approaches“, in *2022 International Interdisciplinary Humanitarian Conference for Sustainability (IIHC)*, Nov. 2022, S. 1276–1281. doi: 10.1109/IIHC55949.2022.10060147. Verfügbar unter: https://ieeexplore.ieee.org/document/10060147. [Zugegriffen: 13. September 2024]

[40] S. Thakare, S. Chavan, und P. Chawan, „Software Testing Strategies and Techniques“, Apr. 2012, Verfügbar unter: https://www.researchgate.net/publication/316513369_Software_Testing_Strategies_and_Techniques. [Zugegriffen: 28. Februar 2025]

[41] S. Jan, S. T. U. Shah, Z. Johar, Y. Shah, und F. Khan, „An Innovative Approach to Investigate Various Software Testing Techniques and Strategies“, Jan. 2016, Verfügbar unter: https://www.researchgate.net/publication/303280520_An_Innovative_Approach_to_Investigate_Various_Software_Testing_Techniques_and_Strategies. [Zugegriffen: 28. Februar 2025]

[42] „Welcome to ISTQB®“, *ISTQB not-for-profit association*. Verfügbar unter: http://istqb-frontend-staging.s3-website.us-east-2.amazonaws.com/. [Zugegriffen: 8. März 2024]
[43] W. Chan, „Product Risk Management: Testing and Warranties“, in *Encyclopedia of Quantitative Risk Analysis and Assessment*, John Wiley & Sons, Ltd, 2008. doi: 10.1002/9780470061596.risk0112. Verfügbar unter: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470061596.risk0112. [Zugegriffen: 13. September 2024]
[44] „Study of the Software Development Life Cycle and the Function of Testing | IEEE Conference Publication | IEEE Xplore“. Verfügbar unter: https://ieeexplore.ieee.org/document/10060231. [Zugegriffen: 9. Juli 2024]
[45] F. Witte, „Testautomatisierung und die Zukunft des Testens“, in *Konzeption und Umsetzung automatisierter Softwaretests*, Wiesbaden: Springer Fachmedien Wiesbaden, 2023, S. 251–259. doi: 10.1007/978-3-658-42661-3_31. Verfügbar unter: https://link.springer.com/10.1007/978-3-658-42661-3_31. [Zugegriffen: 9. März 2024]
[46] Atlassian, „What is Continuous Integration“, *Atlassian*. Verfügbar unter: https://www.atlassian.com/continuous-delivery/continuous-integration. [Zugegriffen: 9. März 2024]
[47] Atlassian, „Continuous deployment“, *Atlassian*. Verfügbar unter: https://www.atlassian.com/continuous-delivery/software-testing/continuous-deployment. [Zugegriffen: 9. März 2024]
[48] N. S. Gill, „Factors affecting effective software quality management revisited“, *SIGSOFT Softw Eng Notes*, Bd. 30, Nr. 2, S. 1–4, März 2005, doi: 10.1145/1050849.1050862. Verfügbar unter: https://doi.org/10.1145/1050849.1050862. [Zugegriffen: 13. September 2024]
[49] „Git - Book“. Verfügbar unter: https://git-scm.com/book/en/v2. [Zugegriffen: 14. September 2024]
[50] G. Biggs, „Applying regression testing to software for robot hardware interaction“, in *2010 IEEE International Conference on Robotics and Automation*, Mai 2010, S. 4621–4626. doi: 10.1109/ROBOT.2010.5509206. Verfügbar unter: https://ieeexplore.ieee.org/document/5509206. [Zugegriffen: 13. September 2024]
[51] H. Lv, Z. Pang, M. Xiao, und G. Yang, „Hardware-in-the-Loop Simulation for Evaluating Communication Impacts on the Wireless-Network-Controlled Robots“, in *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, Okt. 2022, S. 1–6. doi: 10.1109/IECON49645.2022.9968471. Verfügbar unter: https://ieeexplore.ieee.org/document/9968471. [Zugegriffen: 9. Juli 2024]
[52] A. Gotlieb, M. Mossige, und H. Spieker, „Constraint-Guided Test Execution Scheduling: An Experience Report at ABB Robotics“, in *Computer Safety, Reliability, and Security*, J. Guiochet, S. Tonetta, und F. Bitsch, Hrsg., Cham: Springer Nature Switzerland, 2023, S. 64–72. doi: 10.1007/978-3-031-40923-3_6. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-031-40923-3_6. [Zugegriffen: 28. Februar 2025]
[53] S. Liang, „A paradigm of automatic ICT testing system development in practice“, in *2020 International Conference on Advanced Robotics and Intelligent Systems (ARIS)*, Taipei, Taiwan: IEEE, Aug. 2020, S. 1–6. doi:

10.1109/ARIS50834.2020.9205770. Verfügbar unter: https://ieeexplore.ieee.org/document/9205770/. [Zugegriffen: 16. Juli 2024]

[54] A. Mali, G. Singh, und H. S. Gill, „Robotic Arm Based Display Touch Test Automation“, in *2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, Chennai, India: IEEE, Mai 2023, S. 1–7. doi: 10.1109/ACCAI58221.2023.10201044. Verfügbar unter: https://ieeexplore.ieee.org/document/10201044/. [Zugegriffen: 16. Juli 2024]

[55] O. O. Ivoylov, A. G. Kochetov, O. V. Lyang, und K. Libon, „Cost analysis as the basis for management decisions in automation of enzyme immunoassays for HIV infection“, *HIV Infect. Immunosuppr. Disord.*, Bd. 14, Nr. 1, Art. Nr. 1, Mai 2022, doi: 10.22328/2077-9828-2022-14-1-78-89. Verfügbar unter: https://hiv.bmoc-spb.ru/jour/article/view/701. [Zugegriffen: 10. März 2024]

[56] E. Efe, M. Özcan, und H. Hakli, „Building and Cost Analysis of an Industrial Automation System using Industrial Robots and PLC Integration“, *Avrupa Bilim Ve Teknol. Derg.*, Nr. 28, Art. Nr. 28, Nov. 2021, doi: 10.31590/ejosat.972290. Verfügbar unter: https://dergipark.org.tr/en/pub/ejosat/issue/64234/972290. [Zugegriffen: 10. März 2024]

[57] R. Khankhoje, „Robotic Process Automation (RPA) Towards Automation Testing“, *Int. J. Softw. Eng. Appl.*, Bd. 15, Nr. 1, S. 09–16, Jan. 2024, doi: 10.5121/ijsea.2024.15102. Verfügbar unter: https://airconline.com/ijsea/V15N1/15124ijsea02.pdf. [Zugegriffen: 10. März 2024]

[58] W. A. Gruver, „Intelligent robotics in manufacturing, service, and rehabilitation: an overview“, *IEEE Trans. Ind. Electron.*, Bd. 41, Nr. 1, S. 4–11, Feb. 1994, doi: 10.1109/41.281602. Verfügbar unter: https://ieeexplore.ieee.org/document/281602. [Zugegriffen: 14. September 2024]

[59] K.-I. Kim, Y. I. Son, und P. B. S. Kim, „Construction of small humanoids with a new joint actuator module“, in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, Apr. 2004, S. 4510-4514 Vol.5. doi: 10.1109/ROBOT.2004.1302428. Verfügbar unter: https://ieeexplore.ieee.org/document/1302428. [Zugegriffen: 14. September 2024]

[60] N. Demir, P. Demircioglu, und I. Bogrekci, „ADVANCING INDUSTRY 4.0 WITH ROS: A CASE STUDY ON AUTONOMOUS MOBILE ROBOT TECHNOLOGICAL ADVANCEMENTS“, *Int. J. 3D Print. Technol. Digit. Ind.*, Bd. 8, Apr. 2024, doi: 10.46519/ij3dptdi.1366132. Verfügbar unter: https://www.researchgate.net/publication/380121529_ADVANCING_INDUSTRY_40_WITH_ROS_A_CASE_STUDY_ON_AUTONOMOUS_MOBILE_ROBOT_TECHNOLOGICAL_ADVANCEMENTS. [Zugegriffen: 28. Februar 2025]

[61] F. Storiale, E. Ferrentino, und P. Chiacchio, „Robot-Agnostic Interaction Controllers Based on ROS“, *Appl. Sci.*, Bd. 12, Nr. 8, Art. Nr. 8, Jan. 2022, doi: 10.3390/app12083949. Verfügbar unter: https://www.mdpi.com/2076-3417/12/8/3949. [Zugegriffen: 14. September 2024]

[62] S. Susanto, E. Priono, und R. Analia, „Establishing ROS on Humanoid Soccer Robot-BarelangFC Software System“, *J. INTEGRASI*, Bd. 13, Nr. 2, S. 113–121, Okt. 2021, doi: 10.30871/ji.v13i2.3333. Verfügbar unter: https://jurnal.polibatam.ac.id/index.php/JI/article/view/3333. [Zugegriffen: 14. September 2024]

[63] O. El Beggar, B. Bousetta, und T. Gadi, „Generating methods signatures from transition state diagram: A model transformation approach“, in *2012 Colloquium in Information Science and Technology*, Okt. 2012, S. 4–9. doi:

10.1109/CIST.2012.6388054. Verfügbar unter: https://ieeexplore.ieee.org/document/6388054. [Zugegriffen: 14. September 2024]
[64] T. O. Meservy und K. D. Fenstermacher, „Transforming software development: an MDA road map“, *Computer*, Bd. 38, Nr. 9, S. 52–58, Sep. 2005, doi: 10.1109/MC.2005.316. Verfügbar unter: https://ieeexplore.ieee.org/document/1510571. [Zugegriffen: 13. September 2024]
[65] V. Kulkarni und S. Reddy, „Separation of concerns in model-driven development“, *IEEE Softw.*, Bd. 20, Nr. 5, S. 64–69, Sep. 2003, doi: 10.1109/MS.2003.1231154. Verfügbar unter: https://ieeexplore.ieee.org/document/1231154. [Zugegriffen: 13. September 2024]
[66] D. S. Cruzes, N. B. Moe, und T. Dybå, „Communication between Developers and Testers in Distributed Continuous Agile Testing“, in *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, Aug. 2016, S. 59–68. doi: 10.1109/ICGSE.2016.27. Verfügbar unter: https://ieeexplore.ieee.org/document/7577420. [Zugegriffen: 13. September 2024]
[67] J. John, A. Ghosal, T. Margaria, und D. Pesch, „DSLs and Middleware Platforms in a Model-Driven Development Approach for Secure Predictive Maintenance Systems in Smart Factories“, in *Leveraging Applications of Formal Methods, Verification and Validation*, T. Margaria und B. Steffen, Hrsg., Cham: Springer International Publishing, 2021, S. 146–161. doi: 10.1007/978-3-030-89159-6_10. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-030-89159-6_10. [Zugegriffen: 28. Februar 2025]
[68] S. Pichidtienthum, P. Pugsee, und N. Cooharojananone, „Developing Module Generation for Odoo Using Concept of Low-Code Development Platform and Automation Systems“, in *2021 IEEE 8th International Conference on Industrial Engineering and Applications (ICIEA)*, Apr. 2021, S. 529–533. doi: 10.1109/ICIEA52957.2021.9436754. Verfügbar unter: https://ieeexplore.ieee.org/document/9436754. [Zugegriffen: 13. September 2024]
[69] C. Ponsard und V. Ramon, „Survey of automation practices in model-driven development and operations“, in *Proceedings of the Fourth International Workshop on Bots in Software Engineering*, in BotSE ’22. New York, NY, USA: Association for Computing Machinery, Oktober 2022, S. 14–17. doi: 10.1145/3528228.3528405. Verfügbar unter: https://doi.org/10.1145/3528228.3528405. [Zugegriffen: 14. September 2024]
[70] S. Vale *u. a.*, „A Model-driven Approach for Knowledge-based Engineering of Industrial Digital Twins“, in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Västerås, Sweden: IEEE, Okt. 2023, S. 13–23. doi: 10.1109/MODELS58315.2023.00015. Verfügbar unter: https://ieeexplore.ieee.org/document/10343670/. [Zugegriffen: 17. Juli 2024]
[71] I. Guevara, S. Ryan, A. Singh, C. Brandon, und T. Margaria, „Edge IoT Prototyping Using Model-Driven Representations: A Use Case for Smart Agriculture“, *Sensors*, Bd. 24, Nr. 2, S. 495, Jan. 2024, doi: 10.3390/s24020495. Verfügbar unter: https://www.mdpi.com/1424-8220/24/2/495. [Zugegriffen: 17. Juli 2024]
[72] T. Kos, M. Mernik, und T. Kosar, „A Tool Support for Model-Driven Development: An Industrial Case Study from a Measurement Domain“, *Appl. Sci.*, Bd. 9, Nr. 21, S. 4553, Okt. 2019, doi: 10.3390/app9214553. Verfügbar unter: https://www.mdpi.com/2076-3417/9/21/4553. [Zugegriffen: 17. Juli 2024]

[73] M. M. Moe und University of Computer Studies, Hpa-An, Kayin State, Myanmar, „Comparative Study of Test-Driven Development TDD, Behavior-Driven Development BDD and Acceptance Test–Driven Development ATDD“, *Int. J. Trend Sci. Res. Dev.*, Bd. Volume-3, Nr. Issue-4, S. 231–234, Juni 2019, doi: 10.31142/ijtsrd23698. Verfügbar unter: https://www.ijtsrd.com/computer-science/other/23698/comparative-study-of-test-driven-development-tdd-behavior-driven-development-bdd-and--acceptance-test–driven-development-atdd/myint-myint-moe. [Zugegriffen: 20. Juli 2024]

[74] S.-P. Ma, Y.-A. Chen, Y.-J. Guo, und Y.-S. Su, „Semi-Automated Behavior-Driven Testing for the Web Front-Ends“, in *2023 IEEE International Conference on e-Business Engineering (ICEBE)*, Nov. 2023, S. 225–230. doi: 10.1109/ICEBE59045.2023.00048. Verfügbar unter: https://ieeexplore.ieee.org/document/10356167. [Zugegriffen: 20. Juli 2024]

[75] T. Couto, S. Marczak, und F. Gomes, „On the Understanding of How to Measure the Benefits of Behavior-Driven Development Adoption: Preliminary Literature Results from a Grey Literature Study“, in *Proceedings of the XIX Brazilian Symposium on Software Quality*, in SBQS '20. New York, NY, USA: Association for Computing Machinery, März 2021, S. 1–7. doi: 10.1145/3439961.3440000. Verfügbar unter: https://doi.org/10.1145/3439961.3440000. [Zugegriffen: 5. Februar 2024]

[76] F. Xue, J. Wu, und T. Zhang, „Learning-Replay Based Automated Robotic Testing for Mobile App“, *Mob. Inf. Syst.*, Bd. 2022, Nr. 1, S. 1084602, 2022, doi: 10.1155/2022/1084602. Verfügbar unter: https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/1084602. [Zugegriffen: 14. September 2024]

[77] M. Abo Galila, *Mobile App-Entwicklung: Low-/No-Code oder klassische Entwicklung*. 2022. Verfügbar unter: http://pub.fh-campuswien.ac.at/obvfcwhsacc/7942603. [Zugegriffen: 28. Februar 2025]

[78] F. Challakhi, *Entwicklung einer Entscheidungsfindungshilfe für die Nutzung von Low Code/No Code anhand von Microsoft Power Platform*. 2023. Verfügbar unter: http://pub.fh-campuswien.ac.at/obvfcwhsacc/9144899. [Zugegriffen: 28. Februar 2025]

[79] A. Büscher, D. Schilberg, und L. Wiegert, „The Use of Low-Code During a Skill Shortage“, gehalten auf der ASME 2022 International Mechanical Engineering Congress and Exposition, American Society of Mechanical Engineers Digital Collection, Feb. 2023. doi: 10.1115/IMECE2022-95505. Verfügbar unter: https://dx.doi.org/10.1115/IMECE2022-95505. [Zugegriffen: 14. September 2024]

[80] „Applied Sciences | Free Full-Text | Combining Rigorous Requirements Specifications with Low-Code Platforms to Rapid Development Software Business Applications“. Verfügbar unter: https://www.mdpi.com/2076-3417/12/19/9556. [Zugegriffen: 14. September 2024]

[81] H. A. Alsaadi, D. T. Radain, M. M. Alzahrani, W. F. Alshammari, D. Alahmadi, und B. Fakieh, „Factors that affect the utilization of low-code development platforms: survey study“, *Rev. Rom. Informatică Și Autom.*, Bd. 31, Nr. 3, S. 123–140, Sep. 2021, doi: 10.33436/v31i3y202110. Verfügbar unter: https://rria.ici.ro/en/vol-31-no-3-2021/factors-that-affect-the-utilization-of-low-code-development-platforms-survey-study/. [Zugegriffen: 14. September 2024]

[82] G. Juhás, L. Molnár, A. Juhásová, M. Ondrišová, M. Mladoniczky, und T. Kováčik, „Low-code platforms and languages: the future of software development“, in *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Okt. 2022, S. 286–293. doi: 10.1109/ICETA57911.2022.9974697. Verfügbar unter: https://ieeexplore.ieee.org/document/9974697. [Zugegriffen: 14. September 2024]

[83] R. Sanchis, Ó. García-Perales, F. Fraile, und R. Poler, „Low-Code as Enabler of Digital Transformation in Manufacturing Industry“, *Appl. Sci.*, Bd. 10, Nr. 1, S. 12, Dez. 2019, doi: 10.3390/app10010012. Verfügbar unter: https://www.mdpi.com/2076-3417/10/1/12. [Zugegriffen: 16. Juli 2023]

[84] „full.pdf“. Verfügbar unter: https://pub.fh-campuswien.ac.at/obvfcwhsacc/content/titleinfo/7942603/full.pdf. [Zugegriffen: 20. Juli 2024]

[85] H. Dar, M. I. Lali, H. Ashraf, M. Ramzan, T. Amjad, und B. Shahzad, „A Systematic Study on Software Requirements Elicitation Techniques and its Challenges in Mobile Application Development“, *IEEE Access*, Bd. 6, S. 63859–63867, 2018, doi: 10.1109/ACCESS.2018.2874981. Verfügbar unter: https://ieeexplore.ieee.org/document/8513829. [Zugegriffen: 20. Juli 2024]

[86] D. P. Losada, J. L. Fernández, E. Paz, und R. Sanz, „Distributed and Modular CAN-Based Architecture for Hardware Control and Sensor Data Integration“, *Sensors*, Bd. 17, Nr. 5, Art. Nr. 5, Mai 2017, doi: 10.3390/s17051013. Verfügbar unter: https://www.mdpi.com/1424-8220/17/5/1013. [Zugegriffen: 14. September 2024]

[87] „Automatic code generation in practice: experiences with embedded robot controllers | Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences“. Verfügbar unter: https://dl.acm.org/doi/10.1145/2993236.2993247. [Zugegriffen: 14. September 2024]

[88] K. Yuan, L. Ding, M. Abdelfattah, und Z. J. Wang, „LiCaS3: A Simple LiDAR–Camera Self-Supervised Synchronization Method“, *IEEE Trans. Robot.*, Bd. 38, Nr. 5, S. 3203–3218, Okt. 2022, doi: 10.1109/TRO.2022.3167455. Verfügbar unter: https://ieeexplore.ieee.org/document/9770125. [Zugegriffen: 14. September 2024]

[89] „Entwicklung einer Entscheidungsfindungshilfe fÃ¼r die Nutzung von Low Code_No Code anhand von Microsoft Power Platform.pdf“.

[90] H. Algestam, M. Offesson, und L. Lundberg, „Using components to increase maintainability in a large telecommunication system“, in *Ninth Asia-Pacific Software Engineering Conference, 2002.*, Dez. 2002, S. 65–73. doi: 10.1109/APSEC.2002.1182976. Verfügbar unter: https://ieeexplore.ieee.org/document/1182976. [Zugegriffen: 21. Juli 2024]

[91] J. X. Silva, M. Lopes, G. Avelino, und P. Santos, „Low-code and No-code Technologies Adoption: A Gray Literature Review“, in *Proceedings of the XIX Brazilian Symposium on Information Systems*, Maceió Brazil: ACM, Mai 2023, S. 388–395. doi: 10.1145/3592813.3592929. Verfügbar unter: https://dl.acm.org/doi/10.1145/3592813.3592929. [Zugegriffen: 27. Oktober 2024]

[92] S. S. Bhattacharyya und S. Kumar, „Study of deployment of “low code no code” applications toward improving digitization of supply chain management“, *J. Sci. Technol. Policy Manag.*, Bd. 14, Nr. 2, S. 271–287, März 2023, doi:

10.1108/JSTPM-06-2021-0084. Verfügbar unter: https://www.emerald.com/insight/content/doi/10.1108/JSTPM-06-2021-0084/full/html. [Zugegriffen: 27. Oktober 2024]

[93] A. Iung *u. a.*, „Systematic mapping study on domain-specific language development tools“, *Empir. Softw. Eng.*, Bd. 25, Nr. 5, S. 4205–4249, Sep. 2020, doi: 10.1007/s10664-020-09872-1. Verfügbar unter: https://link.springer.com/10.1007/s10664-020-09872-1. [Zugegriffen: 18. Juli 2023]

[94] M. Fowler, „A Pedagogical Framework for Domain-Specific Languages“, *IEEE Softw.*, Bd. 26, Nr. 4, S. 13–14, Juli 2009, doi: 10.1109/MS.2009.85. Verfügbar unter: https://ieeexplore.ieee.org/document/5076452. [Zugegriffen: 21. Juli 2024]

[95] Q. Wang und G. Gupta, „Rapidly prototyping implementation infrastructure of domain specific languages: a semantics-based approach“, in *Proceedings of the 2005 ACM symposium on Applied computing*, in SAC '05. New York, NY, USA: Association for Computing Machinery, März 2005, S. 1419–1426. doi: 10.1145/1066677.1066998. Verfügbar unter: https://doi.org/10.1145/1066677.1066998. [Zugegriffen: 21. Juli 2024]

[96] N. Sushkov und S. Zykov, „Message system refactoring using DSL“, in *2009 5th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR)*, Okt. 2009, S. 153–158. doi: 10.1109/CEE-SECR.2009.5501172. Verfügbar unter: https://ieeexplore.ieee.org/document/5501172. [Zugegriffen: 21. Juli 2024]

[97] L. Chun, G. Gugang, und Y. Jun, „Domain coverage metric for validation“, in *2005 6th International Conference on ASIC*, Okt. 2005, S. 957–960. doi: 10.1109/ICASIC.2005.1611467. Verfügbar unter: https://ieeexplore.ieee.org/document/1611467. [Zugegriffen: 21. Juli 2024]

[98] Q. Ma, M. Kaczmarek-Heß, und S. de Kinderen, „Validation and verification in domain-specific modeling method engineering: an integrated life-cycle view“, *Softw. Syst. Model.*, Bd. 22, Nr. 2, S. 647–666, Apr. 2023, doi: 10.1007/s10270-022-01056-3. Verfügbar unter: https://doi.org/10.1007/s10270-022-01056-3. [Zugegriffen: 21. Juli 2024]

[99] S. Lolong und A. I. Kistijantoro, „Domain Specific Language (DSL) development for desktop-based database application generator“, in *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*, Juli 2011, S. 1–6. doi: 10.1109/ICEEI.2011.6021604. Verfügbar unter: https://ieeexplore.ieee.org/document/6021604. [Zugegriffen: 21. Juli 2024]

[100] M. Bačíková und J. Porubän, „DSL-driven generation of Graphical User Interfaces“, *Open Comput. Sci.*, Bd. 4, Nr. 4, S. 204–221, Dez. 2014, doi: 10.2478/s13537-014-0210-9. Verfügbar unter: https://www.degruyter.com/document/doi/10.2478/s13537-014-0210-9/html. [Zugegriffen: 14. September 2024]

[101] S. Winberg, L. Mohapi, und M. Inggs, „OptiSDR — A domain specific language to improve developer productivity for software defined radio“, in *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, Aug. 2016, S. 20–26. doi: 10.1109/EmergiTech.2016.7737304. Verfügbar unter: https://ieeexplore.ieee.org/document/7737304. [Zugegriffen: 14. September 2024]

[102] S. Thummalapenta, S. Sinha, N. Singhania, und S. Chandra, „Automating test automation“, in *2012 34th International Conference on Software Engineering*

(ICSE), Juni 2012, S. 881–891. doi: 10.1109/ICSE.2012.6227131. Verfügbar unter: https://ieeexplore.ieee.org/document/6227131. [Zugegriffen: 14. September 2024]

[103] „Adapting Keyword driven test automation framework to IEC 61131-3 industrial control applications using PLCopen XML | IEEE Conference Publication | IEEE Xplore“. Verfügbar unter: https://ieeexplore.ieee.org/document/7005185. [Zugegriffen: 14. September 2024]

[104] M. M, „Test Automation of Protection and Control IED Manager using Robot Framework“, *Int. J. Res. Appl. Sci. Eng. Technol.*, Bd. 7, Nr. 6, S. 364–370, Juni 2019, doi: 10.22214/ijraset.2019.6062. Verfügbar unter: https://www.ijraset.com/fileserve.php?FID=23083. [Zugegriffen: 14. September 2024]

[105] „Test Case Analysis with Keyword-Driven Testing Approach on Angkasa Website Using Katalon Studio Tools | Ultimatics : Jurnal Teknik Informatika“. Verfügbar unter: https://ejournals.umn.ac.id/index.php/TI/article/view/2391. [Zugegriffen: 14. September 2024]

[106] I. K. Raharjana, F. Harris, und A. Justitia, „Tool for Generating Behavior-Driven Development Test-Cases“, *J. Inf. Syst. Eng. Bus. Intell.*, Bd. 6, Nr. 1, Art. Nr. 1, Apr. 2020, doi: 10.20473/jisebi.6.1.27-36. Verfügbar unter: https://e-journal.unair.ac.id/JISEBI/article/view/18082. [Zugegriffen: 14. September 2024]

[107] „Low Code Development Platform Market | Size, Share, Growth, Trends | Industry Analysis | Forecast 2026“. Verfügbar unter: https://www.technavio.com/report/low-code-development-platform-market-industry-analysis

[108] N. Prinz, C. Rentrop, und M. Huber, „Low-Code Development Platforms – A Literature Review“, Nr. AMCIS 2021 Proceedings. 2, 2021, Verfügbar unter: https://aisel.aisnet.org/amcis2021/adv_info_systems_general_track/adv_info_systems_general_track/2

[109] M. A. Al Alamin, S. Malakar, G. Uddin, S. Afroz, T. B. Haider, und A. Iqbal, „An empirical study of developer discussions on low-code software development challenges“, *Proc. - 2021 IEEEACM 18th Int. Conf. Min. Softw. Repos. MSR 2021*, S. 46–57, Mai 2021, doi: 10.1109/MSR52588.2021.00018. Verfügbar unter: https://www.researchgate.net/publication/352805735_An_Empirical_Study_of_Developer_Discussions_on_Low-Code_Software_Development_Challenges. [Zugegriffen: 28. Februar 2025]

[110] K. Rokis und M. Kirikova, „Challenges of Low-Code/No-Code Software Development: A Literature Review“, *Lect. Notes Bus. Inf. Process.*, Bd. 462 LNBIP, S. 3–17, 2022, doi: 10.1007/978-3-031-16947-2_1/COVER. Verfügbar unter: https://link.springer.com/chapter/10.1007/978-3-031-16947-2_1

[111] D. Dahlberg, „Developer Experience of a Low-Code Platform: An exploratory study“, Verfügbar unter: https://umu.diva-portal.org/smash/get/diva2:1485348/FULLTEXT01.pdf. [Zugegriffen: 28. Februar 2025]

[112] G. Juhas, L. Molnar, A. Juhasova, M. Ondrisova, M. Mladoniczky, und T. Kovacik, „Low-code platforms and languages: the future of software development“, gehalten auf der 2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia: IEEE, Okt. 2022, S. 286–293. doi: 10.1109/ICETA57911.2022.9974697. Verfügbar unter: https://ieeexplore.ieee.org/document/9974697/. [Zugegriffen: 30. September 2024]

[113] D. Marinković und Z. Ž. Avramović, „Software Platforms Based on the Principles of Graphic Design, Automatic Command Generation and Visual Programming“, *JITA - J. Inf. Technol. Appl. Banja Luka - APEIRON*, Bd. 22, Nr. 2, Okt. 2021, doi: 10.7251/JIT2102110M. Verfügbar unter: http://doisrpska.nub.rs/index.php/jita/article/view/8186. [Zugegriffen: 30. September 2024]

[114] D. Aveiro, V. Freitas, E. Cunha, F. Quintal, und Y. Almeida, „Traditional vs. low-code development: comparing needed effort and system complexity in the NexusBRaNT experiment“, gehalten auf der 2023 IEEE 25th Conference on Business Informatics (CBI), Prague, Czech Republic: IEEE, Juni 2023, S. 1–10. doi: 10.1109/CBI58679.2023.10187470. Verfügbar unter: https://ieeexplore.ieee.org/document/10187470/. [Zugegriffen: 30. September 2024]

[115] L. C. Santos, A. Santos, F. N. Santos, und A. Valente, „A Case Study on Improving the Software Dependability of a ROS Path Planner for Steep Slope Vineyards“, *Robotics*, Bd. 10, Nr. 3, Art. Nr. 3, Sep. 2021, doi: 10.3390/robotics10030103. Verfügbar unter: https://www.mdpi.com/2218-6581/10/3/103. [Zugegriffen: 28. Juli 2024]

[116] F. Erich, A. Saksena, G. Biggs, und N. Ando, „Design and Development of a Physical Integration Testing Framework for Robotic Manipulators“, in *2019 IEEE/SICE International Symposium on System Integration (SII)*, Jan. 2019, S. 602–607. doi: 10.1109/SII.2019.8700444. Verfügbar unter: https://ieeexplore.ieee.org/document/8700444. [Zugegriffen: 28. Juli 2024]

[117] A. Papavasileiou, G. Michalos, und S. Makris, „Quality control in manufacturing – review and challenges on robotic applications“, *Int. J. Comput. Integr. Manuf.*, Bd. 0, Nr. 0, S. 1–37, doi: 10.1080/0951192X.2024.2314789. Verfügbar unter: https://doi.org/10.1080/0951192X.2024.2314789. [Zugegriffen: 28. Juli 2024]

[118] C. Poggi, „End-of-Line test automation using a low cost 6-DOFs robotic arm and ROS.“, März 2020. Verfügbar unter: https://www.semanticscholar.org/paper/End-of-Line-test-automation-using-a-low-cost-6-DOFs-Poggi/1892fb1ed27dffae603f2e133f7476a843ab019e. [Zugegriffen: 28. Juli 2024]

[119] P.-C. Huang und A. K. Mok, „A Case Study of Cyber-Physical System Design: Autonomous Pick-and-Place Robot“, *2018 IEEE 24th Int. Conf. Embed. Real-Time Comput. Syst. Appl. RTCSA*, S. 22–31, Aug. 2018, doi: 10.1109/RTCSA.2018.00012. Verfügbar unter: https://ieeexplore.ieee.org/document/8607230/. [Zugegriffen: 28. Juli 2024]

[120] K. A. Khan, R. R. Konda, und J.-C. Ryu, „ROS-based control for a robot manipulator with a demonstration of the ball-on-plate task“, *Adv. Robot. Res.*, Bd. 2, Nr. 2, S. 113–127, Juni 2018, doi: 10.12989/ARR.2018.2.2.113. Verfügbar unter: https://doi.org/10.12989/ARR.2018.2.2.113. [Zugegriffen: 28. Juli 2024]

[121] G. Walck, U. Cupcic, T. Duran, und V. Perdereau, „A Case Study of ROS Software Re-usability for Dexterous In-Hand Manipulation“, *J. Softw. Eng. Robot.*, Mai 2014, Verfügbar unter: https://www.researchgate.net/publication/340535369_A_Case_Study_of_ROS_Software_Re-usability_for_Dexterous_In-Hand_Manipulation. [Zugegriffen: 28. Februar 2025]

[122] R. Brouzos, K. Panayiotou, E. Tsardoulias, und A. Symeonidis, „A Low-Code Approach for Connected Robots“, *J. Intell. Robot. Syst.*, Bd. 108, Juni 2023, doi: 10.1007/s10846-023-01861-y. Verfügbar unter: https://www.researchgate.net/publication/371698186_A_Low-Code_Approach_for_Connected_Robots. [Zugegriffen: 28. Februar 2025]

[123] S. Han, M. Kim, und H. S. Park, „Open Software Platform for Robotic Services“, *IEEE Trans. Autom. Sci. Eng.*, Bd. 9, Nr. 3, S. 467–481, Juli 2012, doi: 10.1109/TASE.2012.2193568. Verfügbar unter: https://ieeexplore.ieee.org/document/6200387. [Zugegriffen: 28. Juli 2024]
[124] D. Humeniuk, F. Khomh, und G. Antoniol, „A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems“. arXiv, 22. März 2022. Verfügbar unter: http://arxiv.org/abs/2203.12138. [Zugegriffen: 28. September 2024]
[125] C. Crick, G. Jay, S. Osentoski, und O. C. Jenkins, „ROS and Rosbridge: roboticists out of the loop“, *Proc. Seventh Annu. ACMIEEE Int. Conf. Hum.-Robot Interact.*, S. 493–494, März 2012, doi: 10.1145/2157689.2157846. Verfügbar unter: https://dl.acm.org/doi/10.1145/2157689.2157846. [Zugegriffen: 28. Juli 2024]
[126] *Experteninterviews und qualitative Inhaltsanalyse*. Verfügbar unter: https://link.springer.com/book/9783531172385. [Zugegriffen: 7. Oktober 2024]
[127] C. G. M. Vera, M. V. O. Vicente, I. L. A. Vera, M. V. A. Alexander, und H. F. B. Vera, „Low/No-code development platforms and the future of software developers“, *Minerva*, Bd. 1, Nr. Special, Art. Nr. Special, Dez. 2022, doi: 10.47460/minerva.v1iSpecial.76. Verfügbar unter: https://minerva.autanabooks.com/index.php/Minerva/article/view/76. [Zugegriffen: 1. November 2024]
[128] J. X. Silva, M. Lopes, G. Avelino, und P. Santos, „Low-code and No-code Technologies Adoption: A Gray Literature Review“, in *Proceedings of the XIX Brazilian Symposium on Information Systems*, in SBSI ’23. New York, NY, USA: Association for Computing Machinery, Juni 2023, S. 388–395. doi: 10.1145/3592813.3592929. Verfügbar unter: https://doi.org/10.1145/3592813.3592929. [Zugegriffen: 1. November 2024]
[129] P. Tiwari, S. Rajendran, und K. S, „Automatic Performance Verification of Industrial Gateway using Python Framework“, in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Juli 2023, S. 1–7. doi: 10.1109/ICCCNT56998.2023.10307262. Verfügbar unter: https://ieeexplore.ieee.org/abstract/document/10307262. [Zugegriffen: 15. September 2024]
[130] P. Mayring, „Qualitative Content Analysis“, *Forum Qual. Sozialforschung Forum Qual. Soc. Res.*, Bd. Vol 1, S. No 2 (2000): Qualitative Methods in Various Disciplines I: Psychology, Juni 2000, doi: 10.17169/FQS-1.2.1089. Verfügbar unter: http://www.qualitative-research.net/index.php/fqs/article/view/1089. [Zugegriffen: 7. Oktober 2024]
[131] K. Fischer, „Das Governance-Konzept – Entwicklungslinien und Verwendungskontexte“, in *Corporate Sustainability Governance*, in Theorie und Praxis der Nachhaltigkeit. Wiesbaden: Springer Fachmedien Wiesbaden, 2017, S. 85–134. doi: 10.1007/978-3-658-18049-2_3. Verfügbar unter: http://link.springer.com/10.1007/978-3-658-18049-2_3. [Zugegriffen: 3. Januar 2025]

11 Anhang

11.1 Übersicht verwendeter Hilfsmittel

Im Rahmen dieser Arbeit wurde das generative KI-Tool ChatGPT[28] als unterstützendes Hilfsmittel eingesetzt. Die Verwendung beschränkte sich dabei auf folgende Bereiche:

- **Strukturelle Unterstützung und Inspiration bei Textpassagen:**
  ChatGPT wurde genutzt, um Vorschläge für die Strukturierung von Textinhalten zu erhalten. Die generierten Texte dienten dabei als Inspiration und wurden von mir gründlich überprüft, sprachlich und inhaltlich überarbeitet sowie in meinen eigenen Worten neu formuliert. Es wurden keine Textpassagen ohne substanzielle Anpassung übernommen.
- **Datenvisualisierung:**
  Für die grafische Darstellung von Daten wurde ChatGPT verwendet, um entweder direkt Diagramme zu erstellen oder um Python-Code zu generieren, der die Visualisierung ermöglichte. Diese Visualisierungen wurden nach Erstellung von mir geprüft und bei Bedarf angepasst, um die Ergebnisse präzise darzustellen.

Die Endbearbeitung aller Inhalte und Grafiken wurde von mir durchgeführt, sodass mein gestalterischer Einfluss überwiegt und die Arbeit in ihrer Gesamtheit meiner Eigenleistung entspricht.

[28] https://chatgpt.com

11.2 Expert:inneninterview 1 Fragebogen

1. In welchem Altersbereich befinden Sie sich?

   Wählen Sie alle zutreffenden Antworten aus.

   - [ ] 20-30
   - [ ] 30-40
   - [ ] 40-50
   - [ ] 50+

2. In welchem Fachgebiet arbeiten Sie und welche Rolle haben Sie in diesem Bereich?

3. Wie lange arbeiten Sie bereits in diesem Fachgebiet?

   Wählen Sie alle zutreffenden Antworten aus.

   - [ ] =< 5 Jahre
   - [ ] =< 10 Jahre
   - [ ] =< 15 Jahre
   - [ ] =< 20 Jahre
   - [ ] > 20 Jahre

Defintion Fachexperten:
Personen, die Expertenwissen in einem spezifischen Anwendungsgebiet oder einer Branche haben, aber nicht notwendigerweise über umfangreiche Programmierkenntnisse verfügen.

6. Welche Erfahrungen haben Sie bereits mit vereinfachten Softwareentwicklungsmethoden (z.B. BDD, DSLs, Low-Code, No-Code) gemacht?

4. Wie viele Jahre Berufserfahrung haben Sie insgesamt im Bereich IT?

Wählen Sie alle zutreffenden Antworten aus.

- [ ] =< 5 Jahre
- [ ] =< 10 Jahre
- [ ] =< 15 Jahre
- [ ] =< 20 Jahre
- [ ] > 20 Jahre

5. Haben Sie eine technische oder IT-spezifische Ausbildung oder Weiterbildung? (Mehrfachauswahl möglich)

Wählen Sie alle zutreffenden Antworten aus.

- [ ] Ja (Universität)
- [ ] Ja (HTL o.ä.)
- [ ] Ja (Kurs, Weiterbildung oder Sonstiges)
- [ ] Nein

Fragen zu vereinfachten Softwareentwicklungsmethoden

Definition vereinfachte Softwareentwicklungsmethoden:
Ansatz, der verschiedene Technologien und Methoden wie Low-Code- und No-Code-Plattformen, Behavior Driven Development und Domain Specific Languages umfasst, um die Softwareentwicklung effizienter, zugänglicher und benutzerfreundlicher zu gestalten, indem sie die technischen Barrieren reduziert und eine stärkere Ausrichtung auf Geschäfts- und Benutzeranforderungen ermöglicht.
Defintion Fachexperten:
Personen, die Expertenwissen in einem spezifischen Anwendungsgebiet oder einer Branche haben, aber nicht notwendigerweise über umfangreiche Programmierkenntnisse verfügen.

6. Welche Erfahrungen haben Sie bereits mit vereinfachten Softwareentwicklungsmethoden (z.B. BDD, DSLs, Low-Code, No-Code) gemacht?

7. Welchen Umfang haben diese Ansätze in Ihrem Projekt(en)?

Markieren Sie nur ein Oval.

	1	2	3	4	
gering	○	○	○	○	sehr hoch

8. Welche zentralen Anforderungen und Kriterien sollten solche Entwicklungsansätze Ihrer Meinung nach erfüllen?

9. Auf Basis Ihrer Erfahrungen, welche Besonderheiten des besagten Ansatzes sind in der Testautomatisierung maßgeblich für eine potentielle Optimierung?

10. Welche Herausforderungen oder Limitierungen könnten bei der Verwendung dieser vereinfachten Entwicklungsansätze auftreten?

11. Wie sieht die Zusammenarbeit, bzw. Integration von Fachexperten in ihrem Projekt aus?

12. Wie bewerten Sie die Bedeutung der Integration von Fachexperten, sodass diese ohne Programmierkenntnisse automatisierte Tests erstellen und warten können?

Markieren Sie nur ein Oval.

	1	2	3	4	
gering	○	○	○	○	sehr hoch

13. Welche Vorteile sehen Sie in der Einbindung von Fachexperten?

14. Welche Auswirkungen hat die Integration von Fachexperten auf die Wartung und Weiterentwicklung von Testfällen?

15. Welche Vorteile sehen Sie in der Verwendung dieser vereinfachten Entwicklungsansätze ir Vergleich zu herkömmlichen Ansätzen (ohne Einbezug von Fachexperten) ?

16. Auf einer Skala von 1 bis 4, wie würden Sie die Effizienz dieser anforderungsspezifischen Entwicklungsansätze im Vergleich zu herkömmlichen Ansätzen unter Annahme das diese eingesetzt werden können, bewerten?

Markieren Sie nur ein Oval.

	1	2	3	4	
sehr ineffizient	○	○	○	○	sehr effizient

11.3 Transkripte für Expert:inneninterview 1

11.3.1 ExpertIn 1

Interviewer: Super, dann legen wir mal los. Welches Safari-Fenster siehst du gerade?
ExpertIn: Ich glaube, ich sehe deinen Fragebogen, wenn das das Richtige sein soll.

Interviewer: Genau, perfekt. Dann können wir gleich mit den demografischen Fragen beginnen. Welches Alterskästchen würdest du ankreuzen?
ExpertIn: Das vorletzte, bitte.

Interviewer: Und dein Fachbereich bzw. deine Rolle?
ExpertIn: Softwareentwicklung, Software Engineer trifft es ganz gut.

Interviewer: Wie viele Jahre professionelle Erfahrung?
ExpertIn: Mehr als 20 Jahre würde ich sagen.

Interviewer: In Ordnung. Und die Ausbildung?
ExpertIn: Ja, da habe ich Uni-Abschluss. Keine HTL.

Interviewer: Alles klar. Dann kurz zur Erklärung, wieso wir über „vereinfachte Softwareentwicklungsmethoden“ sprechen: Darunter fallen Low-Code, No-Code, Behavior Driven Development, Domain-specific Languages und alles, was es Nicht-Programmierer:innen erleichtern soll, sich an der Softwareentwicklung oder Testautomatisierung zu beteiligen. Mit Fachexpert:innen sind eben Leute gemeint, die das Domänenwissen haben, aber keine oder nur wenig Programmierfahrung.
ExpertIn: Verstanden.

Interviewer: Prima. Dann zu deinen Erfahrungen: Hast du in deinen Projekten solche vereinfachten Methoden schon intensiv genutzt? Zum Beispiel DSLs, BDD, Low-Code?
ExpertIn: DSLs auf jeden Fall. Die anderen Konzepte kenne ich eher theoretisch, aber nicht im Sinne von professionell täglich genutzt. Bei DSLs habe ich schon praktische Erfahrung.

Interviewer: Verstehe. Dann können wir uns in den nächsten Fragen vor allem auf DSLs konzentrieren. In welchem Umfang hattest du das in Projekten im Einsatz?
ExpertIn: Recht hoch. Wir haben etwa im Kontext der deutschen Gesundheitskarte damit Testfälle beschrieben. Das war also sehr umfangreich im Einsatz.

Interviewer: Kannst du kurz schildern, wie das technisch oder organisatorisch ausgesehen hat?
ExpertIn: Wir haben eine eigene DSL entwickelt, speziell um Testfälle fachlich zu beschreiben. Also eher eine vereinfachte Programmiersprache, die genau auf das Domänenwissen zugeschnitten war. Der Test-Interpreter konnte dann diese DSL ausführen. So konnten Fachexpert:innen, die zwar etwas Technikaffinität haben, aber keine professionellen Entwickler:innen sind, Tests formulieren und automatisieren.

Interviewer: Also wart ihr im QA-Bereich oder wie war das?
ExpertIn: Genau, Qualitätssicherung. Die Fachleute hatten sehr tiefgehendes Wissen über die Vorgaben rund um die Gesundheitskarte. Da kam die DSL ins Spiel, um dieses Fachwissen ohne großen Programmieraufwand in Tests zu gießen.

Interviewer: Was würdest du sagen, inwiefern war dieser Ansatz relevant für die Menge der automatisierten Tests oder dafür, dass das, was der Kunde wollte, auch wirklich abgedeckt wurde?
ExpertIn: Das Fachwissen war so groß, dass es ohne die Expert:innen gar nicht möglich gewesen wäre, die richtigen Testfälle zu bauen. Der DSL-Ansatz hat das vereinfacht und beschleunigt, weil wir alles fachlich passgenau gestalten konnten. Gleichzeitig ließ sich damit eine große Testfallvielfalt erreichen.

Interviewer: Verstehe. Was waren denn so Herausforderungen oder Limitierungen bei diesem DSL-Ansatz?
ExpertIn: Na ja, das Warten der DSL selbst ist nicht zu unterschätzen. Man muss laufend Scope-Entscheidungen treffen: Wie speziell oder wie allgemein soll die DSL sein? Zu speziell – dann fehlen einem vielleicht wichtige Funktionen. Zu allgemein – dann ist es fast schon wieder eine normale Programmiersprache. Außerdem braucht’s Personal, das sowohl ein wenig Technik als auch das Domänenwissen hat.

Interviewer: Wie habt ihr die Zusammenarbeit mit den Fachexpert:innen konkret organisiert?
ExpertIn: Die Testscripts lagen in einem Versionierungssystem, ähnlich wie Sourcecode. Es gab dazu eine Dokumentation in einem Wiki, in der die Fachbefehle erklärt waren. Die Fachexpert:innen konnten so neue Testfälle schreiben und einchecken, und das System, das die DSL interpretierte, lief dann gegen die Zielanwendung.

Interviewer: Wie wichtig findest du es generell, dass Fachexpert:innen in so einem Prozess mitwirken und dass sie – zumindest theoretisch – sogar ohne klassische Programmierkenntnisse Tests erstellen und warten können?

ExpertIn: Das ist sehr stark kontextabhängig. In unserem Fall war das Domänenwissen so technisch, dass die meisten Expert:innen sowieso eine gewisse IT-Affinität hatten. In einem anderen Kontext kann es natürlich wichtig sein, eine noch einfachere Plattform anzubieten. Ich würde sagen, die Möglichkeit, dass Fachexpert:innen selbst Hand anlegen können, ist eine gute Sache, sofern der Aufwand für die Plattform sich lohnt. Ich würde das auf einer Skala in etwa bei zwei ansiedeln, weil es nicht immer zwingend notwendig ist. Aber klar, es kann Vorteile haben, vor allem wenn Fachpersonal massenhaft verfügbar ist.

Interviewer: Welche Vorteile siehst du in der Einbindung dieser Leute?
ExpertIn: Ohne sie fehlt dir das Wissen, was genau getestet werden muss. Sie kennen die Abläufe. Also man vermeidet, dass man nur aus Entwicklerperspektive testet und das Fachliche zu kurz kommt.

Interviewer: Und in Bezug auf die Wartung der Tests? Ist das dann nicht super, wenn die Fachleute später Änderungen selbst einpflegen können?
ExpertIn: Ja, das wäre ein Idealfall. In der Praxis scheitert's manchmal am Projektmanagement oder am Budget, sobald das Projekt in den Wartungsmodus geht. Aber wenn man es richtig aufsetzt, dann sollten die Fachexpert:innen die Tests tatsächlich selbst weiterpflegen können.

Interviewer: Gut. Abschließend: Wenn du das bewerten müsstest – wie effizient sind solche „vereinfachten" Ansätze im Vergleich zu herkömmlichen (wo die Entwickler:innen alles machen) auf einer Skala von 1 bis 4?
ExpertIn: Ich würde sagen eine 3. Kann sehr effizient sein, wenn man's richtig aufsetzt. Eine 4 wäre, wenn es universell immer klappt. Das tut es nicht. Es ist sehr stark abhängig vom Projekt, vom Personal, vom Budget, von der Komplexität. Aber prinzipiell kann es viel bringen.

Interviewer: Alles klar. Danke für das Gespräch. Hättest du noch etwas zu ergänzen?
ExpertIn: Nein, ich glaube, das war es. Hoffe, es hilft dir weiter.

Interviewer: Auf jeden Fall, danke dir. Dann stoppe ich die Aufzeichnung jetzt.

11.3.2 ExpertIn 2

Interviewer: Hallo! Bevor wir mit dem eigentlichen Interview beginnen, gebe ich dir kurz einen Überblick über den Ablauf: Wir machen zwei Expert:inneninterviews – eines vor dem Konzept und eines nach dem Konzept. Heute ist das erste Interview dran. Es geht darum, das Thema sowie die Anforderungen an eine Low-Code-Plattform für robotergestützte Testautomatisierung zu untermauern und ein Fundament für die Konzeptphase zu legen.

ExpertIn: Alles klar, gerne. Dann leg einfach los.

Interviewer: Super. Magst du dich kurz vorstellen? Also in einem groben Altersbereich, dein aktuelles berufliches Umfeld und wie du damit zu unserem Thema stehst?

ExpertIn: Ich bin zwischen 40 und 50 Jahre alt und arbeite als Software Engineer und Researcher, aktuell mit einer Rolle als Technical Lead in einem Software-Projekt. Insgesamt bin ich seit ungefähr 20 Jahren in der IT tätig und habe mein Studium an der TU beziehungsweise allgemein an einer Universität abgeschlossen.

Interviewer: Perfekt. Dann gehen wir schon zu den themenspezifischen Fragen: Unsere Arbeit behandelt sogenannte *spezifikations- bzw. problemenspezifische Entwicklungsansätze* – das umfasst zum Beispiel Behavior Driven Development (BDD) mit Gherkin/Cucumber, domänenspezifische Sprachen (DSLs) oder Low-Code-Plattformen. Welche Erfahrungen hast du in diese Richtung?

ExpertIn: Ich habe in einem früheren Projekt viel mit BDD gearbeitet, konkret mit Gherkin und Cucumber. Das lief in einem Java-/Selenium-Umfeld. Wir haben große Enterprise-Applikationen damit getestet und sehr viel Wert auf End-to-End-Tests gelegt. Die BDD-Tests wurden in natürlicher Sprache geschrieben, was dem Kunden half, die Testfälle zu verstehen. Aufgezeichnete Tests oder Rekord-Replay-Ansätze kamen manchmal auch zum Einsatz, wobei ich das eher als „No-Code“ bezeichnen würde.

Interviewer: In welchem Umfang habt ihr denn diesen Ansatz im Projekt eingesetzt?

ExpertIn: Das war tatsächlich ein wesentlicher Bestandteil unserer Akzeptanztests. Wir hatten unzählige automatisierte Tests, die in Untergruppen aufgeteilt wurden. Die Ausführung all dieser Tests dauerte sehr lange, zum Teil mehrere Tage. Aber wir brauchten diese massive Testabdeckung, weil es sich um eine sehr komplexe Enterprise-Lösung handelte.

Interviewer: Welche zentralen Anforderungen oder Kriterien haben dazu geführt, dass ihr euch für Gherkin/Cucumber bzw. allgemein diesen BDD-Ansatz entschieden habt?

ExpertIn: Zum einen wollte man eine solide Regressionstestabdeckung sicherstellen. Zum anderen war es wichtig, dass die Tests von Kundenseite verstanden werden – also möglichst in natürlicher Sprache. Auch die Lücke zwischen Unit-Tests und dem eigent-

lichen End-to-End-Verhalten war zu groß. Durch BDD konnten wir früh im Konzeptionsprozess Testfälle in natürlicher Sprache definieren, was dem Kunden wie auch uns bei der Qualitätssicherung geholfen hat.

Interviewer: Du hast gerade schon ein paar Vorteile genannt. Was würdest du noch als besondere Vorteile solcher anforderungsspezifischen Ansätze hervorheben – gerade im Vergleich zu rein manuellen Tests oder herkömmlichen Automatisierungsansätzen (z.B. nur Unit-Tests)?

ExpertIn: Erstens bindet man den Fachbereich viel früher und konkreter ein. Die Testfälle werden in einer Art geschrieben, die auch Nicht-Programmierer:innen verstehen können. Zweitens überlegt man sich die Tests schon in der Konzeptions- oder Analysephase und kommt damit potenziellen Missverständnissen früh auf die Spur. Und drittens lässt sich dadurch – bei richtiger Pflege – ein hoher Automatisierungsgrad erzielen, sodass man bei komplexen Anwendungen schneller merkt, wenn irgendwo etwas „Fundamentales“ kaputtgegangen ist.

Interviewer: Stichwort „richtige Pflege“: Welche Erfahrungen hast du denn bei der Wartung und Weiterentwicklung dieser Tests gemacht?

ExpertIn: Wartung ist das größte Thema. Wenn man einmal zu schreiben aufhört oder die Testfälle nicht kontinuierlich anpasst, werden die Tests sehr schnell veraltet und fallen aus. In einem Großprojekt, das über viele Jahre lief, hatte das Team irgendwann hohe Fluktuation. Manche Testfälle wurden bei Änderungen an der Software einfach nicht mehr angepasst. Dann hat man irgendwann viele „rote“ Tests, die im Daily Build nur stören, und deaktiviert sie vielleicht. Damit geht allerdings die Abdeckung wieder verloren.

Interviewer: Verstehe. Wie würdest du die Bedeutung einschätzen, Fachexpert:innen tatsächlich so einzubinden, dass sie ohne Programmierkenntnisse selbst Tests schreiben und warten können?

ExpertIn: Ich würde die Bedeutung sehr hoch einschätzen. Idealerweise definiert der Fachbereich die Testfälle gleich mit oder gemeinsam mit Analysten. Dann hat man diesen „sanity check“: Passt das so, wie ihr es braucht? Gleichzeitig muss man sich klar sein, dass das auch viel Zeit und Aufwand bedeutet. Manche Kunden wollen das nicht übernehmen oder haben gar nicht die Kapazitäten, sich so intensiv in den Prozess einzubringen. Aber wenn es gelingt, hat man definitiv Vorteile.

Interviewer: Wo siehst du technische oder organisatorische Limits? Also ab wann stößt ein solcher Ansatz an seine Grenzen?

ExpertIn: Bei sehr vielen externen Schnittstellen und komplexen Mocks kann BDD mühsam werden. Wenn man unzählige Systeme angebunden hat, ist das Ganze extrem wartungsintensiv. Auch Themen wie Performance- oder Security-Tests löst man nicht über BDD in natürlicher Sprache. Daneben ist es eben ein sozial-organisatorisches Problem: Wenn niemand mehr die Verantwortung für die Pflege übernimmt, ist das Tooling egal – die Tests verrotten über die Zeit.

Interviewer: Letzte Skalenfrage zum Abschluss: Unter der Annahme, dass es ein Projekt mit starkem Domänenwissen ist – wie effizient ist so ein spezifikationsgetriebener, anforderungsspezifischer Ansatz im Vergleich zu rein herkömmlichem, eher manuell geprägtem Vorgehen? Auf einer Skala von 1 (niedrig) bis 4 (hoch).

ExpertIn: Ich tendiere da auf jeden Fall zu 4, also sehr hoch. Natürlich investiert man am Anfang mehr Zeit für die Test- und Konzeptionsarbeit, aber wenn das Projekt so angelegt ist, dass viele Abstimmungen nötig sind, zahlt sich das am Ende aus – vorausgesetzt, man bleibt in der Wartung dran.

11.3.3 ExpertIn 3

Interviewer:
Okay, dann starten wir einmal offiziell. Ich gebe einen kurzen Überblick und eine kurze Einführung zu meiner Diplomarbeit: Heute ist das erste von zwei geplanten Expert:inneninterviews für meine Diplomarbeit. Genauer gesagt ist es das zweite Interview in der ersten Interviewrunde, denn letzte Woche hatte ich bereits ein erstes Gespräch und konnte dabei den Fragebogen leicht anpassen.

Die Richtung meiner Diplomarbeit geht in den Bereich Software-Qualitätssicherung. Ich arbeite auch bei der [Firma] in diesem Bereich und bin dort zuständig für Projekte wie [Projekt] – das ist aber für die Diplomarbeit nicht so wesentlich.

Der Fokus der Diplomarbeit liegt auf der **Konzeption und Umsetzung einer Low-Code-Plattform für einen robotergestützten Testansatz**. Das bedeutet, dass wir Fachexpert:innen – also Personen ohne große Programmierkenntnisse – in die Testautomatisierung einbeziehen wollen. Zum Beispiel, indem sie mithilfe eines Roboters Testfälle eigenständig automatisieren können.

Heute möchten wir einmal grundsätzlich den Ansatz untermauern und die Anforderungen erheben, die für eine Konzeption so einer Low-Code- oder No-Code-Plattform relevant sind. Vielleicht noch zur Konkretisierung: Wir haben Hardware (einen Roboter), die wir nicht virtualisieren können oder wollen, aber das ist nicht der Hauptpunkt – es geht vor allem darum, eine Plattform bereitzustellen, die den Schritt weg vom reinen Code hin zu einer natürlicheren Sprache oder Bedienung schafft, sodass auch FachAnwender:innen Testfälle definieren können. Der Roboter dient eher als Ausblick, wofür man das Ganze konkret einsetzen könnte.

Ist das soweit klar?

ExpertIn:
Ja, das passt für mich. Können wir gerne loslegen.

Interviewer:
Super. Dann beginnen wir kurz mit den demografischen Fragen. Ich habe hier etwas auf dem Bildschirm geteilt. Sie meinten, Sie sind 38 Jahre alt, richtig?

ExpertIn:
Genau, ich bin 38.

Interviewer:
Und wie könnten wir Ihr Fachgebiet zusammenfassen? Also einerseits fachlich, andererseits technisch?

ExpertIn:
Fachlich bin ich im E-Health-Bereich tätig, genauer in der Gesundheitstelematik mit besonderem Fokus auf den deutschen Markt. Technisch bin ich Softwarearchitekt und „unfreiwilliger" Projektleiter.

Interviewer:
Wie lange schon ungefähr in diesem Bereich?

ExpertIn:
Ich bin insgesamt seit rund 20 Jahren in der IT. Davon natürlich nicht alles E-Health, aber eben viel Projekterfahrung in verschiedenen Domänen.

Interviewer:
Okay, und was Ihre Ausbildung angeht – Universität? Oder HTL vorher?

ExpertIn:
Ich bin von einer AHS gekommen, habe dann ein Studium an der Uni gemacht. Keine HTL.

Interviewer:
Kommen wir zum Fachteil. Wir haben den Sammelbegriff „vereinfachte Softwareentwicklungsmethoden" gewählt. Darunter fallen Low-Code, No-Code, Behavior Driven Development (BDD) und domänenspezifische Sprachen (DSLs). Das Ziel ist, die Entwicklung effizienter und zugänglicher zu machen, insbesondere für Personen ohne Programmierkenntnisse, also Fachexpert:innen.

Welche Erfahrungen haben Sie mit solchen Ansätzen? Was fällt Ihnen dazu ein?

ExpertIn:
Das ist schon eine Weile her, aber ich hole kurz aus: Ich habe mich historisch und auch im Rahmen meiner Dissertation mit **alten Low-Code-Ansätzen** oder eher 4GL (Fourth Generation Languages) beschäftigt, die in Banken und Versicherungen intensiv verwendet wurden. Das war mehr Richtung: „FachAnwender:innen sollen selbst entwickeln können." Hat sich in komplexen Projekten nie durchgesetzt, da es doch sehr beschränkt war.

DSLs hatte ich öfter in Projekten, zum Beispiel in einer Mainframe-Ablöse am Flughafen. Da ging es um eine Art DSL für Testdaten – man konnte damit Flüge in bestimmten Status anlegen, was sehr hilfreich für die Business-Analysten war. Ziel war, Lesbarkeit und Verständlichkeit zu erhöhen, nicht unbedingt, dass reine Fachleute selber programmieren.

BDD mit Gherkin habe ich immer mal wieder evaluiert, aber nie in großem Stil eingeführt, weil der Overhead zu hoch war. Oft waren Entwickler:innen schneller, wenn sie direkt in Code gearbeitet haben, als ein BDD-Framework samt „glue code" zu pflegen. Das ist meine persönliche Erfahrung dazu.

Interviewer:
Verstehe. In welchem Umfang hatten diese Ansätze in Ihren Projekten denn tatsächlich Bedeutung?

ExpertIn:
Meistens eher gering. Ich würde sagen eher 1 oder maximal 2 auf einer Skala von 1 bis 4.

Interviewer:
Okay. Dann zur nächsten Frage: Bei den Projekten, in denen Sie etwas in der Richtung eingesetzt haben – was waren da die ausschlaggebenden Anforderungen, die zu so einem vereinfachten Entwicklungsansatz geführt haben?

ExpertIn:
Zum Beispiel beim Mainframe-Ablöseprojekt: weg von reiner Low-Level-Technik, mehr benutzerfreundliche Prozessabbildung, um Fach-Workflows leichter abzubilden. Bei der Test-DSL am Flughafen war das Ziel, die Testdatenerstellung rasch und abstrakt zu gestalten. Da war es dann sehr hilfreich, einen Flug einfach in einer fachlichen DSL zu beschreiben, statt zig Datenbank-Tabellen oder APIs.

Was BDD angeht, da hätte es die Anforderung geben müssen, dass wirklich Fachexpert:innen selbst Tests schreiben wollen. Das war aber nicht der Fall, bzw. der Aufwand, so ein Framework zu erstellen und zu warten, war höher, als wenn Entwickler:innen es direkt programmieren.

Interviewer:
Okay, verstehe. Nächste Frage: Wenn man so einen vereinfachten Ansatz wählt, gerade in Bezug auf Automatisierung – was ist Ihrer Meinung nach entscheidend für eine gelungene Optimierung?

ExpertIn:
Es kommt immer auf die Zielgruppe an. Ich persönlich mag DSLs *in* der Programmiersprache selbst, zum Beispiel Java, weil man keinen großen Extra-Overhead hat. Geht es aber darum, echte Nicht-Programmierer:innen einzubinden, kann man zu BDD greifen, sofern man sie nur für Abnahmetests oder End-to-End-Tests nutzt. Für Unit-Tests wäre das völlig überdimensioniert.

Außerdem braucht man ein Bewusstsein dafür, dass so etwas Wartung kostet und in den Entwicklungsprozess eingebettet sein muss. Man kann es nicht einfach einmal aufsetzen und dann liegen lassen.

Interviewer:
Stichwort Wartung. Wie schätzen Sie den Wartungsaufwand solcher Methoden ein, verglichen mit einem rein technischen Ansatz?

ExpertIn:
Definitiv höherer Aufwand, weil man eine zusätzliche Abstraktionsebene einführt. Man muss das *reinpreisen*, sprich: Lohnt es sich, dass man Fachexpert:innen einbindet und dafür mehr Wartungsaufwand hat? Das kann sinnvoll sein, wenn der Nutzen – etwa bessere Verständigung, schnellere Klärung von fachlichen Fragen – sehr hoch ist. Es ist aber schwer zu verallgemeinern, kommt stark aufs Projekt an.

Interviewer:
Ja, absolut. Dann zur nächsten Frage: Was sehen Sie als große Herausforderungen oder Limitierungen bei diesen vereinfachten Ansätzen?

ExpertIn:
Erstens das richtige Abstraktionslevel. Je mehr man abstrahiert, desto mehr „glue code" oder Framework-Logik braucht man. Zweitens die Edge Cases und Fehleranalysen. Wenn etwas schiefläuft, reicht reines Fachwissen meist nicht, da braucht man Entwicklungskompetenz, um Fehler zu debuggen.

Drittens sind reine „Nicht-Techniker" häufig nicht in der Lage, tiefergreifende Testfälle selbst zu formulieren, weil die Komplexität eben bleibt. BDD & Co. helfen eher in puncto Lesbarkeit und Kommunikation als beim Eliminieren von Komplexität.

Interviewer:
Wie lief in den Projekten, von denen Sie sprachen, die Integration von Fachexpert:innen ab?

ExpertIn:
In den meisten Fällen gab es Business-Analysten oder Requirements Engineers. Eine direkte Linie zwischen Fachexpert:innen und Entwickler:innen habe ich selten erlebt, eher eine Vermittlerrolle, die etwas Technik kann und etwas Fachdomäne. Reine Fachexpert:innen sitzen selten im Kern des Entwicklerteams.

Interviewer:
Und würden Sie sagen, Fachexpert:innen ohne Programmierkenntnisse können am Ende tatsächlich selbst Automatisierungsskripte schreiben und warten?

ExpertIn:
Eher nein. Ich sehe die Rolle der Fachexpert:innen darin, *konzeptionell* mitzuarbeiten, Inputs zu liefern, Tests auf einer fachlichen Ebene zu beschreiben. Aber das *eigentliche* Testskript erstellt meist ein Techniker, eventuell mithilfe einer DSL oder BDD. Ganz ohne technische Zwischenrolle klappt es meist nicht.

Interviewer:
Welche Vorteile sehen Sie dennoch bei der Einbindung solcher Fachexpert:innen – egal ob sie nun direkt Skripte tippen oder nicht?

ExpertIn:
Der riesige Vorteil ist, dass sie die Domäne kennen und man damit das **Risiko minimiert**, am realen Bedarf vorbeizuentwickeln. Wer weiß sonst besser, was das System am Ende leisten soll, als diejenigen, die fachlich dafür verantwortlich sind?

Interviewer:
Wie wirkt sich das gegebenenfalls auf die Wartung und Weiterentwicklung der Tests nach der ersten Umsetzung aus?

ExpertIn:
Das Potenzial ist da: Wenn Fachexpert:innen wissen, was geändert wurde, könnten sie Testfälle anpassen oder zumindest definieren. Aber in der Realität verschwinden Fachexpert:innen oft nach Projektende, Budgets sind begrenzt, und dann kümmert sich kaum einer intensiv um den Testbestand. Das ist eher eine organisatorische Schwäche und passiert leider häufig.

Interviewer:
Alles klar. Vorletzte Frage: Die Vorteile dieses vereinfachten Entwicklungsansatzes gegenüber einem herkömmlichen Vorgehen, wo der Kunde vielleicht nur Feedback gibt und das Team alles umsetzt – wie fassen Sie die nochmal zusammen?

ExpertIn:
Es schafft eine bessere Kommunikationsbasis, indem man eine gemeinsame Sprache hat, etwa Gherkin oder eine DSL. Das macht die Tests und die Anforderungen transparenter. Es ersetzt nicht den Dialog mit dem Kunden, kann ihn aber strukturierter machen.

Interviewer:
Und abschließend: Wenn Sie auf einer Skala von 1 (gering) bis 4 (hoch) die Effizienz eines solchen vereinfachten Ansatzes bewerten sollen – unter der Annahme, er wäre passend für das jeweilige Projekt – wo würden Sie sich einordnen?

ExpertIn:
Ich würde in meiner Erfahrung eine 2 geben. Das Potenzial kann zwar hoch sein, aber nur selten wird es ausgeschöpft. Oft ist der Aufwand höher als der direkte Nutzen, wenn die Einbindung von Fachexpert:innen nicht durchgängig gelebt wird oder zu wenig Ressourcen bereitstehen.

Interviewer:
Wunderbar, vielen Dank für Ihre Zeit und die ausführlichen Antworten. Möchten Sie noch etwas hinzufügen, das wir nicht angesprochen haben?

ExpertIn:
Eigentlich nicht. Ich hab’s eh frei erzählt. Ich hoffe, es hilft Ihnen weiter.

11.4 Expert:inneninterview 2 Fragebogen

Feedback zum NoCode-Konzept um Fachexperten ohne Programmierkenntnisse in den Entwicklungsprozess miteinzubeziehen

mit Fokus auf Projekte mit Software und Hardwarekomponenten

1. In welchem Altersbereich befinden Sie sich?

   Wählen Sie alle zutreffenden Antworten aus.

   - [ ] 20-29
   - [ ] 30-39
   - [ ] 40-50
   - [ ] 51+

2. In welchem Fachgebiet arbeiten Sie und welche Rolle haben Sie in diesem Bereich?

3. Wie lange arbeiten Sie bereits in diesem Fachgebiet?

   Wählen Sie alle zutreffenden Antworten aus.

   - [ ] =< 5 Jahre
   - [ ] =< 10 Jahre
   - [ ] =< 15 Jahre
   - [ ] =< 20 Jahre
   - [ ] > 20 Jahre

4. Wie viele Jahre Berufserfahrung haben Sie insgesamt im Bereich IT?

Wählen Sie alle zutreffenden Antworten aus.

- [ ] =< 5 Jahre
- [ ] =< 10 Jahre
- [ ] =< 15 Jahre
- [ ] =< 20 Jahre
- [ ] > 20 Jahre

5. Haben Sie eine technische oder IT-spezifische Ausbildung oder Weiterbildung? (Mehrfachauswahl möglich)

Wählen Sie alle zutreffenden Antworten aus.

- [ ] Ja (Universität)
- [ ] Ja (HTL o.ä.)
- [ ] Ja (Kurs, Weiterbildung oder Sonstiges)
- [ ] Nein

Teil 1/4: Feedback zum Konzept

6. In welcher Phase des Entwicklungsprozesses sehen Sie den Einsatz des Konzepts sinnvoll?

(z. B. während der Anforderungserhebung, parallel zur Entwicklung oder abschließend bei der Abnahme?)

7. Inwieweit stimmen Sie der folgenden Aussage zu:
"Das Konzept, Fachexperten ohne Programmierkenntnisse in den Entwicklungsprozess einzubinden, könnte den Entwicklungsprozess verbessern."

Markieren Sie nur ein Oval.

	1	2	3	4	
stimme überhaupt nicht zu	○	○	○	○	stimme voll und ganz zu

8. Wie stark glauben Sie, dass dieses Konzept die Zusammenarbeit zwischen Fachexperten und Entwicklern verbessert?

Markieren Sie nur ein Oval.

	1	2	3	4	
überhaupt nicht	○	○	○	○	sehr stark

9. Welche konkreten Vorteile sehen Sie in diesem Konzept ?

10. Welche Herausforderungen oder Bedenken haben Sie bezüglich der Umsetzung dieses Konzepts?

Teil 2/4: Vergleich mit traditionellen Ansätzen

11. Welche spezifischen Vorteile oder Nachteile sehen Sie im No-Code Ansatz gegenüber traditionellen Methoden bei der Testautomatisierung?

12. Sehen Sie in komplexen Projekten einen Mehrwert durch die Einbindung von Fachexperten mittels des vorgestellten Konzepts? Bitte erläutern Sie Ihre Meinung

Teil 3/4: Technische Aspekte und Integration

13. Würden Sie in komplexen Projekten Fachexperten mit dem vorgestellten Konzept in den Entwicklungsprozess einbeziehen wollen?

Markieren Sie nur ein Oval.

- ja
- Nein

14. Wie wichtig ist für Sie die Integration des Konzepts in Continuous Integration (CI) Prozesse?

Markieren Sie nur ein Oval.

	1	2	3	4	
nicht wichtig	○	○	○	○	sehr wichtig

Teil 4/4 Potenzial und Zukunftsaussichten

15. Wie wahrscheinlich ist es, dass Sie eine Umsetzung dieses Konzepts in zukünftigen Projekten verwenden würden?

Markieren Sie nur ein Oval.

	1	2	3	4	
sehr unwahrscheinlich	○	○	○	○	sehr wahrscheinlich

16. Welche Bedingungen müssten erfüllt sein, damit Sie das Konzept in Ihrem Arbeitsumfeld einsetzen?

17. Welche zusätzlichen Funktionen oder Verbesserungen würden Sie sich für das Konzept wünschen?

18. Haben Sie weitere Anmerkungen oder Feedback zum vorgestellten Konzept?

11.5 Transkripte für Expert:inneninterview 2

11.5.1 ExpertIn 1

Interviewer:
Hier noch keine konkreten Fragen. Dann würde ich jetzt weiter zum Expert:inneninterview überleiten. Passt das für dich?

ExpertIn:
Ja, passt.

Interviewer:
Dann legen wir einmal los mit den demographischen Fragen. Die erste Frage: Wie ist dein Altersbereich? Welches Kästchen kann ich hier ankreuzen?

ExpertIn:
Passt, da kannst du 30-40 ankreuzen.

Interviewer:
In welchem Fachgebiet arbeitest du und welche Rolle hast du?

ExpertIn:
Ich arbeite in der Qualitätssicherung. Als übergeordneten Begriff kannst du „QS“ oder „Qualitätssicherung“ nehmen. Genauer gesagt in Testmanagement und Testautomatisierung.

Interviewer:
Wie viele Jahre Berufserfahrung hast du in diesem Fachgebiet und insgesamt im Bereich IT und welche Ausbildung hast du gemacht?

ExpertIn:
Beim Fachgebiet kannst du <=5 angeben, bei insgesamt <=10 und Universität als Ausbildung.

Interviewer:
Gut, das waren die demografischen Fragen. Dann kommen wir schon zum eigentlichen Teil: dem Feedback zum Konzept. Bezüglich Entwicklungsprozess – wann würdest du die Fachexpert:innen am ehesten einbinden: schon bei der Anforderungsanalyse oder erst hinterher?

ExpertIn:
Am besten so früh wie möglich, um gleich bei der Spezifikation die nötigen Testschritte anzudenken. In einem agilen Setting kann man das pro Story tun und gleich festlegen, welche Schritte man braucht. Dann entwickelt das Team diese Steps, und wenn es fertig ist, kann der Test in derselben Iteration laufen. Bei Wasserfallprojekten könnte man zumindest in der Spezifikationsphase die Testfälle festhalten, sodass sie hinterher automatisch abgearbeitet werden.

Interviewer: Ok. „Wie weit stimmen Sie der folgenden Aussage zu: Das Konzept, Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess einzubinden, könnte den Entwicklungsprozess verbessern.“ Dabei geht es erst mal nur um das übergeordnete Konzept – nicht zwingend um den konkreten Prototyp. Was würdest du hier für einen Wert geben?

ExpertIn:
Ich würde sagen: Mit etwas Klarstellung, wo genau dieses Konzept ansetzt, stimme ich zu. Unter der Annahme, dass man die Integration in den Entwicklungsprozess vorher klar definiert, würde ich so auf eine 3 auf deiner Skala gehen.

Interviewer:
Verstanden. Nächste Frage: Wie stark glaubst du, dass dieses Konzept die Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen verbessern kann, indem es eine neue Kommunikationsebene schafft?

ExpertIn:
Vielleicht eine 3. Ich sehe da durchaus Potenzial, weil durch das gemeinsame Definieren der Tests viele Missverständnisse früh aufgedeckt werden können. So werden Anforderungen klarer.

Interviewer:
Welche konkreten Vorteile siehst du in dem Konzept? Also was könnte so ein Ansatz leisten?

ExpertIn:
Ein großer Vorteil ist, dass die Fach- und die Entwicklerseite eine einheitliche Sprache benutzen, beispielsweise über gemeinsam definierte Testschritte oder eine No-Code-Oberfläche. Dadurch werden Missverständnisse reduziert und man hat quasi eine schriftliche Dokumentation der fachlichen Anforderungen. Es kann zudem frühzeitig abgesteckt werden, wie die Abnahme aussieht, weil die Tests das ja schon abbilden.

Interviewer:
Und welche Herausforderungen oder Bedenken hast du bei so einer Umsetzung?

ExpertIn:
Da sehe ich mehrere Punkte:

Valide Inputdaten: Die Fachperson muss wissen, welche konkreten Daten an einer bestimmten Stelle nötig sind. Woher soll er wissen, welche IDs existieren, welche Werte gültig sind usw.?
Kombination von Testschritten: Wie verhindert man, dass eine Fachperson falsche Schritte kombiniert oder logische Zwischenschritte weglässt, sodass der Test technisch nicht durchführbar ist?
Duplikate: Wenn mehrere Fachexpert:innen dieselben Schritte anfordern, könnte es zu doppelten Steps kommen. Man braucht also einen Mechanismus, um das zu vermeiden.
Versionierung und Wartung: Wenn sich Abläufe im System ändern oder es mehrere Software-Versionen gibt, müssen Tests entsprechend angepasst oder parallel geführt

werden können. Wie man das dem Fachexpert:innen nahebringt, ist eine Herausforderung.
Abgrenzung zu BDD: Warum sollte man diesen Ansatz einer No-Code-Plattform nutzen, anstatt klassisches BDD (z. B. Gherkin) oder Keyword-driven Tests zu verwenden? Wo ist der Mehrwert?

Dann gleich eine Anschlussfrage: Wie unterscheidet sich das Ganze von klassischen BDD-Ansätzen?

Interviewer:
Aus meiner Sicht setzt das Konzept auf eine noch niedrigere Einstiegshürde. Bei BDD muss man zumindest die Gherkin-Syntax kennen und teilweise auch verstehen, wie Steps in Code hinterlegt werden. Ein No-Code-Konzept könnte für fachliche Leute intuitiver sein, weil sie sich alles per Klick zusammenbauen – ohne etwas schreiben zu müssen. Gleichzeitig bietet es eine integrierte Kommunikationsebene zum Entwicklungsteam.

ExpertIn: Ok gut.

Interviewer:
Welche Vor- und Nachteile siehst du in diesem No-Code-Ansatz gegenüber traditionellen Methoden?

ExpertIn:
Vorteile:

- Fachexpert:innen können direkt mitwirken, da sie keine Programmiersprache lernen müssen.
- Potenziell schnellere und klarere Kommunikation über Anforderungen und Tests.
- Man hat eine Art lebende Spezifikation.

Nachteile:

- Die zuvor genannten Probleme wie Versionierung, Duplikate oder falsche Reihenfolgen bleiben eine Herausforderung.
- Die Entkopplung vom Code kann auch zu neuen Missverständnissen führen, etwa wenn nicht klar ist, was technisch möglich ist.
- Ein zusätzliches Tool oder eine Plattform erhöht den Organisationsaufwand.

Interviewer:
Denkst du, dass es in komplexen Projekten mit mehreren Software- und Hardware-Komponenten einen besonderen Mehrwert gibt, Fachexpert:innen so einzubinden?

ExpertIn:
Ja, klar. Gerade bei komplexen Projekten ist das Fachwissen umso wichtiger. Die Fachexpert:innen kennen die Anforderungen, wissen, wie das System eingesetzt werden soll, und können so schon sehr früh sagen, wie Tests aussehen müssen. Außerdem kann es

für die Abnahme nützlich sein: Wenn ein Testfall durch den Fachexpert:innen definiert und grün ist, hat das quasi Abnahmecharakter.

Interviewer:
Würdest du Fachexpert:innen mit diesem Konzept in deinen zukünftigen Projekten einbeziehen?

ExpertIn:
Je nach Projekt würde ich das ernsthaft in Betracht ziehen. Wenn die Rahmenbedingungen stimmen, die Fachexpert:innen verfügbar sind und man diesen Ansatz in den Prozess integriert, ist das durchaus wahrscheinlich.

Interviewer:
Wie wichtig ist die Integration in die CI-Pipeline?

ExpertIn:
Sehr wichtig. Nur so hat man eine durchgängige Automatisierung. Wenn Fachexpert:innen ihre Tests definieren, aber es nicht in der CI läuft, wäre es wieder nur manuell oder halbautomatisch. Dann hätte man keinen echten Mehrwert.

Interviewer:
Wenn du an die Bedingungen denkst, die erfüllt sein müssen, damit dieses Konzept in der Praxis funktioniert – was sind für dich die wichtigsten?

ExpertIn:

- **Kontinuierliche Einbindung der Fachexpert:innen**: Nicht nur am Anfang oder Ende, sondern als Teil des Teams.
- **Klare Prozessdefinition**: Man muss wissen, wann in einem agilen Ablauf z. B. die Testschritte angefordert und umgesetzt werden.
- **Versionierungskonzept**: Gerade bei längeren Projekten müssen die Tests gepflegt werden, ohne dass man ständig alles umwirft.
- **Übersichtlichkeit**: Man braucht Mechanismen, um Duplikate oder falsche Kombinationen von Schritten zu vermeiden.

Interviewer:
Hast du noch weitere Ideen oder Verbesserungsvorschläge für das Konzept?

ExpertIn:
Ja, ein paar Punkte:

- **Review-Prozess**: Neue Testfälle könnten einen „Draft-Status“ haben und müssten von jemandem geprüft werden, bevor sie automatisch laufen.
- **Testfall-Gruppierung**: Man könnte Sammlungen von Testfällen anlegen, um bestimmte Release-Zyklen oder Integrationsprüfungen zu fahren.
- **Meta-Informationen zu den Testschritten**: Wer kann sie wie nutzen, welche Eingaben sind möglich, in welcher Reihenfolge sind Schritte kombinierbar, etc.?

Interviewer:
Alles klar. Das waren im Wesentlichen meine Fragen. Vielen Dank für deine Zeit und das Feedback.

ExpertIn:
Gerne, ich hoffe, das hilft dir weiter.

Interviewer:
Definitiv, danke dir! Dann wäre das Interview an dieser Stelle beendet. Schönen Tag noch!

ExpertIn:
Gleichfalls!

11.5.2 ExpertIn 2

Interviewer: Also zuerst: Wie alt sind Sie? Welches Fachgebiet und welche Rolle haben Sie aktuell in diesem Bereich?

ExpertIn:
40 bis 50 kannst du ankreuzen. Ich arbeite im Bereich Software Engineering und bin darüber hinaus wissenschaftlicher Mitarbeiter. Im Moment bin ich außerdem Technical Lead im Projekt.

Interviewer:
Okay, passt. In Ihrem aktuellen Hauptfachgebiet: Wie viele Jahre Erfahrung haben Sie dort? Wir rechnen hierbei ohne Ausbildung, rein beruflich.

ExpertIn:
Dann sind es in diesem Bereich „kleiner 15“ Jahre.

Interviewer:
Verstanden. Und die allgemeine Berufserfahrung im IT-Bereich, wenn man alles zusammenzählt?

ExpertIn:
Da kommen wir dann auf „kleiner 20“. Würde ich also bei der nächsten Kategorie angeben.

Interviewer:
Alles klar. Wie sieht es mit der höchsten technischen Ausbildung aus?

ExpertIn:
Ich habe einen Universitätsabschluss. Weiterbildungen könnte man ebenfalls anhaken.

Interviewer:
Perfekt, dann kommen wir schon zur ersten Frage, die wir vorhin angesprochen haben: „In welcher Phase des Entwicklungsprozesses sehen Sie das vorgestellte Konzept als besonders sinnvoll an?“

ExpertIn:
Ich habe eher einen iterativen Zugang zum Software Engineering. Das heißt, man kann dieses Konzept eigentlich in allen Phasen einsetzen: zu Beginn in der Planungs- oder Leistungsbeschreibungsphase, während der Implementierung und auch nachher bei Abnahmen oder Regressionstests. Prinzipiell ziehe ich das über den ganzen Software-Lebenszyklus.

Interviewer:
Dann gehen wir weiter zu den quantitativen Fragen. Auf einer Skala von 1 (stimme überhaupt nicht zu) bis 4 (stimme voll zu): „Das Konzept, Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess einzubinden, könnte den Entwicklungsprozess verbessern.“ Wie würden Sie das bewerten?

ExpertIn:
Ich bin generell ein Verfechter davon, die fachlichen DomänenExpert:innen stark einzubinden. Also hier würde ich eine 4 vergeben. Ohne enge Zusammenarbeit mit der Fachseite geht es kaum noch.

Interviewer:
Nächste Frage: „Wie stark glauben Sie, dass dieses Konzept die Zusammenarbeit zwischen Entwickler:innen und Fachexpert:innen verbessern kann?“ Wieder 1 bis 4.

ExpertIn:
Das bewerte ich mit einer 3. Es ist ein gutes Werkzeug, das Zusammenarbeit besser strukturieren kann. Ob es die Kommunikation wirklich verbessert, hängt auch von den menschlichen Faktoren und der Projektorganisation ab. Aber das Tool kann natürlich helfen.

Interviewer:
Alles klar. Welche konkreten Vorteile sehen Sie in dem Konzept?

ExpertIn:
Der Hauptvorteil ist, dass man Leute aus dem Fachbereich stärker dazu bringt, Tests und Testfälle aktiv mitzugestalten. Dadurch hat man aus dem Fachbereich mehr Inputs, was früh zu höherer Qualität führen kann. Man denkt schneller und fundierter über die Use Cases nach. Ein weiterer Vorteil ist die leichtere Bedienbarkeit via GUI oder Klickoberfläche; das senkt die Einstiegshürde gegenüber reinem Code.

Interviewer:
Welche Herausforderungen oder Bedenken haben Sie bei der Umsetzung?

ExpertIn:

- Ich sehe die Gefahr, dass der Fachbereich zwar genau weiß, „was“ er will, aber nicht „wie“ man das in einzelne, sinnvolle Schritte strukturiert. Das ist normalerweise Entwicklungstätigkeit.
- Das Risiko, dass sich ähnliche oder identische Schritte duplizieren, wenn das nicht gut organisiert ist. Man braucht ein Terminologie- und Steps-Konzept, sonst degeneriert das Ganze.
- Versionierung, Variablenhandling und komplexe Abläufe (z.B. API-Sequenzen) müsste das Tool unterstützen, sonst wird’s in der Praxis schnell unübersichtlich.

Interviewer:
Verstanden. Kommen wir zur Frage nach No-Code im Vergleich zu klassischen Ansätzen (wo Entwickler:innen den Testcode schreiben). Was würden Sie als wichtigsten Vorteil im No-Code-Ansatz hervorheben?

ExpertIn:
Der große Vorteil ist die niedrigere Eintrittsschwelle. Fachexpert:innen müssen keinen Code lesen oder schreiben. Sie klicken sich ihre Testfälle zusammen, was ihnen einen schnelleren Zugang verschafft. Ansonsten ähneln die Anforderungen an gute Terminologie und Struktur auch denen bei BDD oder Keyword-Driven-Tests.

Interviewer:
Und wie sehen Sie Nachteile oder Herausforderungen speziell beim No-Code-Ansatz?

ExpertIn:

- Auch No-Code braucht im Hintergrund Entwickler:innen, die die Schritte implementieren. Es ist also nicht „Magie“, dass gar kein Code mehr existiert.
- Das Tool muss professionell integriert sein (CI-Pipeline, Git etc.) und gepflegt werden. Sonst hat man zwar eine schöne Klickoberfläche, aber in der Praxis scheitert es.
- Skalierung und Wiederverwendbarkeit: In größeren Projekten hat man rasch Hunderte von Steps, da muss man streng auf Ordnung achten.

Interviewer:
Würden Sie in komplexen Projekten – also mit mehreren Soft- und Hardwarekomponenten – Fachexpert:innen mittels des vorgestellten Konzepts einbinden?

ExpertIn:
Ja, auf jeden Fall. Gerade in komplexen Projekten bringt das Fachwissen enorm viel. Da macht es Sinn, wenn der Fachbereich möglichst früh und intensiv Testfälle mitentwickelt.

Interviewer:
Gut. Würden Sie das auch wirklich in Ihren nächsten Projekten nutzen wollen?

ExpertIn:
Wenn das Konzept ausgereift ist und die nötigen Features (z.B. Versionierung, Variablen, Verzweigungen) mitbringt, sehe ich da durchaus einen großen Nutzen. Also ja, sehr wahrscheinlich.

Interviewer:
Wie wichtig ist Ihnen die Integration in die CI-Pipeline?

ExpertIn:
Das ist für mich essenziell (würde ich mit 4 bewerten). Denn ohne automatisierte Pipeline fehlen die Vorteile der kontinuierlichen Prüfung. Gerade bei einer DevOps- oder agilen Umgebung muss so etwas in die CI/CD-Landschaft eingebunden sein.

Interviewer:
Welche Bedingungen müssen erfüllt sein, damit Sie das Konzept in Ihrem Arbeitsumfeld einsetzen würden?

ExpertIn:

- Das Tool oder der Ansatz müsste stabil, skalierbar und in unsere Tools wie GitLab, Jira etc. integrierbar sein.
- Eine klare Struktur, damit wir nicht 100 Duplikate bei den Schritten haben.
- Gute Handhabung von Parametern, Variablen und Vor-/Nachbedingungen für realistische Szenarien.

Interviewer:
Haben Sie noch weitere Vorschläge oder Anmerkungen?

ExpertIn:

- Man sollte Vorbedingungen und Nachbedingungen klar trennen, damit der Fachbereich die Abläufe strukturiert.
- Schleifen, Bedingungslogik und Verzweigungen wären sinnvoll, sonst kann man viele reale Szenarien nicht abbilden.
- Die Idee, mithilfe eines Guis den Einstieg zu erleichtern, ist sehr gut. Man sollte aber das Risiko adressieren, dass Fach- und Entwicklungsseite eng kooperieren müssen, um Schrittduplikate zu vermeiden und das Ganze wirklich sauber aufzubauen.

Interviewer:
Alles klar. Dann vielen Dank für die Beantwortung der Fragen und das ausführliche Feedback!

ExpertIn:
Gern geschehen. Ich hoffe, das hilft dir weiter.

Interviewer:
Absolut – vielen Dank und einen schönen Abend noch!

ExpertIn:
Danke, ebenfalls!

11.5.3 ExpertIn 3

Interviewer:
Kommen wir zunächst zu den demographischen Fragen, die wir durchgehen würden. Alter?

ExpertIn:
Vorletzte Kategorie passt.

Interviewer:
In welchem Fachgebiet und in welcher Rolle bist du aktuell tätig?

ExpertIn:
Ich habe gefühlt 70 Fachgebiete (lacht). Aber gut, ich würde sagen „PM“ (Projektmanagement), um es kurz zu machen.

Interviewer:
Okay, passt. Wie viele Jahre Berufserfahrung insgesamt im Bereich?

ExpertIn:
Vorletzte Kategorie hier auch, also eins darunter … passt so.

Interviewer:
Verstanden. Hier könnte man mehrfach auswählen, aber du hast dich für „ganz oben“ entschieden?

ExpertIn:
Ja, nur ganz oben.

Interviewer:
Dann kommen wir schon zur Bewertung des Konzepts, also nicht des Prototyps, sondern des übergeordneten Konzepts. Die erste Frage: „In welcher Phase des Entwicklungsprozesses halten Sie den Einsatz des Konzepts für sinnvoll?“

ExpertIn:
Wahrscheinlich in allen Phasen. Immer dort, wo ein Testfall entsteht oder ein Test durchgeführt werden soll, kann man ansetzen. Und je früher man damit beginnt, desto besser – zumindest gefühlt.

Interviewer:
Alles klar. Dann eine Skalen-Frage: „Das Konzept, Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess einzubinden, könnte den Entwicklungsprozess verbessern.“ Skala von 1 bis 4, wobei 4 = „Stimme voll zu“ ist.

ExpertIn:
Ich würde hier eine 4 geben, also voll und ganz zustimmen.

Interviewer:
Wie stark glaubst du, dass dieses Konzept die Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen verbessern kann? 1 heißt „gar nicht“, 4 heißt „sehr stark“.

ExpertIn:
Das würde ich auf eine 3 setzen.

Interviewer:
Gut. Welche konkreten Vorteile siehst du in dem Konzept?

ExpertIn:

- Man nimmt die Testfallerstellung weg von den klassischen Testern und verlagert sie an die Leute, die wirklich das Fachwissen haben.
- In der Theorie bietet das eine vereinfachte Darstellung der Abläufe und Prozesse.
- Es unterstützt die Kommunikation, indem Fachleute ihre Sicht einbringen.

Ob das in der Realität tatsächlich immer so reibungslos funktioniert, ist eine andere Frage, aber die Idee ist erstmal gut.

Interviewer:
Und welche Herausforderungen oder Bedenken kommen dir in den Sinn?

ExpertIn:

- Letztlich verteilt man die Arbeitslast nur um. Ich glaube nicht, dass man sie reduziert, weil jetzt viele Micro-Interaktionen zwischen Fachleuten und Entwickler:innen nötig werden.
- Ich bin mir auch nicht sicher, ob das Konzept für sehr komplexe Systeme ausreicht, insbesondere ob Leute komplett ohne technischen Background das verstehen.
- Bei vielen Actions/Steps wird es schnell unübersichtlich. Auch sind manche Begrifflichkeiten in so einem Tool zu technisch. Beispiel: Pipeline-Runs oder IDs.
- Ob die Fachperson sich langfristig für die Ergebnisse der Pipeline interessiert, bezweifle ich. Vielleicht braucht es verschiedene Rollen: Fachexperte definiert die Logik, Tester validiert, Entwickler:innen macht den technischen Feinschliff.
- Tests können auch fehlschlagen, weil sie falsch definiert sind – dafür braucht es nochmal jemanden, der den Test fachlich und methodisch prüft.

Interviewer:
Kannst du dir vorstellen, dass man Fachexpert:innen durch Schulungen oder Handson-Demos an dieses Tool heranführt?

ExpertIn:
Ja, sicher. Man könnte z. B. mehr Beispiele im Wiki bereitstellen. Oder mehr Hilfstexte und Tooltips im Tool selbst. Auch eine gründliche Schulung, damit die Leute verstehen, was ein Testfall überhaupt ist, wie man ihn methodisch aufbaut und was er leisten soll. Das würde definitiv helfen.

Interviewer:
Welche Vor- oder Nachteile siehst du im Vergleich zu den traditionellen Methoden der Testautomatisierung, wo meist nur das Entwicklungsteam Tests umsetzt?

ExpertIn:

- **Vorteil**: Man schiebt mehr Verantwortung zu den Fachexpert:innen, was die Testdefinition angeht. Das kann die Entwicklung entlasten und gleichzeitig ist die Fachseite stärker eingebunden.
- **Nachteil**: Ganz allein werden Fachexpert:innen das nicht stemmen können. Man braucht meist noch einen Tester, der methodisch draufschaut, und einen Entwickler:innen für die technischen Details.

Oft weiß eine Fachperson zum Beispiel nicht genau, wie viele Varianten eines Testfalls nötig sind. Viele denken an den Gutfall, vergessen aber Fehlerfälle. Das muss man schulen oder durch einen Testprofi abfangen.

Interviewer:
Alles klar. Glaubst du, dass das gerade bei komplexen Projekten einen deutlichen Mehrwert bringt?

ExpertIn:
Im Prinzip schon, weil man das geballte Wissen aus verschiedenen Fachrichtungen einbeziehen kann. Ein Tool oder Ansatz, wo mehrere Fachexpert:innen ihren Teil in einen Testablauf einbringen, kann sehr nützlich sein. Vor allem für Abnahmetests in Großprojekten.

Interviewer:
Alles klar. Würdest du also in komplexen Projekten Fachexpert:innen mit dem Konzept einbeziehen?

ExpertIn:
Ja, aber nur, wenn es strukturiert abläuft und die Leute die nötige Schulung oder Unterstützung bekommen. Sonst entsteht Chaos.

Interviewer:
Nächste Frage: „Wie wichtig ist die Integration in die CI-Pipeline?“ (Skala 1–4)

ExpertIn:
Aus Sicht der Fachexpert:innen? Ich glaube, das ist nicht wirklich entscheidend für sie. Ich gebe hier eine 1, weil ich nicht glaube, dass sich die meisten Fachexpert:innen um Pipeline-Runs kümmern wollen. Das obliegt eher den Testverantwortlichen und Entwickler:innen.

Interviewer:
Wie wahrscheinlich ist es insgesamt, dass du so ein Konzept in deinen Projekten einsetzen würdest?

ExpertIn:
Wenn es vernünftig umgesetzt ist, würde ich das mit 3 von 4 bewerten.

Interviewer:
Welche Bedingungen müssten deiner Meinung nach erfüllt sein?

ExpertIn:

- Schulung und Hilfestellungen für die Fachexpert:innen (Tooltips, Wiki, Beispiele).
- Gute Rollenverteilung: Fachexperte definiert die Domäne, Tester überprüft die Testlogik, Entwickler:innen sorgt für die technische Einbindung.
- Eine gewisse Visualisierung der Testkomplexität wäre hilfreich: Wenn es 50 Actions gibt, sollte man eine Warnung sehen.
- Man könnte KI einsetzen, um Testschritte vorzuschlagen oder generieren zu lassen.
- Gamification, um Leute zu motivieren, mehr Testfälle zu definieren oder zu validieren.

Interviewer:
Klasse, vielen Dank! Dann wäre das alles von meiner Seite. Danke für deine Zeit!

ExpertIn:
Gerne. Ich muss jetzt leider los, aber ich hoffe, das hat dir geholfen.

Interviewer:
Absolut. Danke dir und bis bald!

ExpertIn:
Bis bald. Ciao!

11.5.4 ExpertIn 4

Interviewer:
Wir haben ein paar demographische Fragen im Vorfeld zu klären. Zunächst: Welches Kästchen darf ich hier bei deinem Alter ankreuzen?

ExpertIn:
Ich bin zwischen 30 und 39 – zum Glück!

Interviewer:
Alles klar. Was gibst du als Fachgebiet an?

ExpertIn:
Ich arbeite in der Architektur, Softwarearchitektur

Interviewer:
Und wie viele Jahre bist du in diesem Fachgebiet tätig? Danach kommt noch die Frage nach deinen IT-Gesamtjahren.

ExpertIn:
Für das Fachgebiet nehmen wir oben 15, bei den IT-Gesamtjahren dann 20.

Interviewer:
Passt. Bei der technischen Ausbildung oder Weiterbildung – was möchtest du hier ankreuzen?

ExpertIn:
Universität reicht. Mehrfachauswahl ist möglich, aber ich bleibe bei Uni.

Interviewer:
Super. Dann kommen wir zur ersten inhaltlichen Frage: „In welcher Phase des Entwicklungsprozesses siehst du den Einsatz des Konzepts als sinnvoll an?“ Also zum Beispiel in der Anforderungserhebung, im Design, in der Implementierung oder abschließend in der Abnahme?

ExpertIn:
Grundsätzlich macht es Sinn, schon während der Anforderungserhebung bzw. Design-Phase darüber nachzudenken, Testfälle zu definieren oder zu skizzieren. Man muss natürlich bedenken, dass sich während der Implementierung noch einiges ändert und man die Tests dann anpassen muss. Aber auf lange Sicht – auch für Abnahme- und Wartungsphasen – halte ich es für sinnvoll, weil es die Regressionstests vereinfacht.

Interviewer:
Verstanden. Dann eine Bewertungsfrage von 1 (stimme gar nicht zu) bis 4 (stimme voll zu): „Das Konzept, Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess einzubinden, könnte den Entwicklungsprozess verbessern.“ Was würdest du wählen?

ExpertIn:
Ich gebe hier eine 3, weil es definitiv hilfreich ist, Fachexpert:innen einzubinden. Gleichzeitig kostet es aber auch Zeit und Ressourcen, und das muss man abwägen.

Interviewer:
Und wie stark glaubst du, dass das Konzept die Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen verbessert, wieder auf einer Skala von 1 bis 4?

ExpertIn:
Ich sage auch 3. Das hängt stark von der Qualität der „Sprache“ ab, die man gemeinsam entwickelt – man kann es ja als eine Art DSL verstehen. Wenn die gut gemacht ist, verbessert das die Zusammenarbeit

Interviewer:
Welche konkreten Vorteile siehst du im Konzept?

ExpertIn:

- Es schafft ein gemeinsames Vokabular in Form einer DSL oder eines Tools, das beide Seiten verstehen.
- Es berücksichtigt den Entstehungsprozess, also dass Steps erweitert und verändert werden können.
- Es erleichtert fachliches Feedback früh und macht Tests für Fachexpert:innen besser lesbar.

Interviewer:
Welche Herausforderungen oder Bedenken siehst du?

ExpertIn:

- Die Welt dreht sich weiter: Wenn sich viel ändert, muss man die Tests ständig nachpflegen.
- Es fehlt teilweise der direkte „Refactoring-Support“ wie in einer reinen Code-Umgebung. Da kann es passieren, dass Tests „kaputtgehen“, weil sich ein Entwickler:innen-Interface ändert.
- Die Traceability zwischen Codeänderungen und Tests kann leiden, wenn die Testsprache zu stark vom Code entkoppelt ist.

Interviewer:
Okay. Jetzt zu den Vor- und Nachteilen eines No-Code-Ansatzes gegenüber traditionellen, codebasierten Methoden (wo üblicherweise das Entwicklerteam die Tests erstellt). Was fällt dir dazu ein?

ExpertIn:

- **Vorteil**: Lesbarkeit und Verständlichkeit steigen. Man kann mehr Stakeholder einbeziehen, die nicht programmieren können oder wollen.
- **Nachteil**: Die Mächtigkeit solcher no-code-basierten Sprachen ist begrenzt. Bei sehr komplexen Fällen kann es unübersichtlich werden oder man kommt an die Grenzen des Tools

Interviewer:
Alles klar. Denkst du, dass gerade in komplexen Projekten – mit Hardware- und Software-Komponenten – ein besonderer Mehrwert entsteht, wenn man Fachexpert:innen so einbindet?

ExpertIn:
Man hat durchaus Vorteile, weil Expert:innenwissen besser eingebracht werden kann. Dennoch bleibt das Problem, dass man das Framework bereitstellen und pflegen muss. Wenn der Wortschatz nicht von Anfang an ausreichend ist, muss man ständig neue Bausteine implementieren. Die Kosten dürfen nicht den Nutzen übersteigen. Aber für die Wartung und für spätere Prüfungen kann es sehr gut sein, weil Fachexpert:innen die Tests ansehen und beurteilen können.

Interviewer:
Wie sieht's speziell mit Abnahmetests aus?

ExpertIn:
Könnte helfen, dass die abnehmende Seite das System besser versteht. Dennoch ist man eingeschränkt durch die bereits definierten Test-Schritte. Auftraggeber möchten oft frei testen – wenn das Tool nur bestimmte Schritte anbietet, fühlt sich das vielleicht eingeschränkt an.

Interviewer:
Verstanden. Würdest du dieses Konzept in deinen komplexen Projekten einsetzen wollen?

ExpertIn:
Ja, tendenziell schon – wenn ein paar Rahmenbedingungen stimmen. Kommt eben aufs Projekt an.

Interviewer:
Welche Rolle spielt dabei die Integration in die CI-Pipeline?

ExpertIn:
Sehr wichtig. Vor allem für kontinuierliches Testen auf unterschiedlichen Code-Ständen. Das ist wohl die größte praktische Herausforderung.

Interviewer:
Dann noch eine Skalenfrage: „Wie wahrscheinlich ist es, dass du in einem zukünftigen Projekt ein solches Konzept verwendest?" (1 = sehr unwahrscheinlich, 4 = sehr wahrscheinlich)

ExpertIn:
Ich würde eine 3 geben.

Interviewer:
Welche Bedingungen müssen erfüllt sein?

ExpertIn:

- Das Projekt-Setup muss passen. Wenn es nur ein simpler Webshop ist, lohnt sich der Aufwand vielleicht nicht.
- Eine gute Integrationsmöglichkeit in unser vorhandenes Issue-Tracking, Source-Control und CI/CD-Setup.
- Echt verfügbare Fachexpert:innen, damit nicht wieder alles die Entwickler:innen übernehmen.

Interviewer:
Hast du noch konkrete Verbesserungsvorschläge oder Anmerkungen?

ExpertIn:

- Markierung für unfertige oder fehlende Steps (z. B. farblich), damit man gleich sieht, was noch nicht lauffähig ist.
- Parameter-Handling: Konsistente Verwaltung von IDs oder Testobjekten, damit es nicht zu Tippfehlern oder Redundanzen kommt. Eher eine Dropdown-Liste oder Konstanten, um beliebig kopierte Fehler zu vermeiden.
- Bessere Log-Anzeige: Wenn ein Test rot ist, sollte man direkt in der visuellen Teststruktur sehen, wo es gekracht hat.
- Man muss die Wartbarkeit und die Weiterentwicklung der Testschritte sehr ernst nehmen – nicht nur anfangs, sondern fortlaufend.
- Die initiale DSL braucht wahrscheinlich gemeinsame Workshops zwischen Entwickler:innen und Fachexpert:innen, sonst verfehlt man schnell die richtige Granularität.

Interviewer:
Danke für dein umfangreiches Feedback! Zum Schluss vielleicht noch: Hast du einen Tipp, wenn man schon in der Frühphase des Projekts startet?

ExpertIn:
Man sollte Zeit für gemeinsame Meetings einplanen. Auf der „grünen Wiese“ eine DSL zu definieren, ist nicht trivial. Beide Seiten, Fach- und Entwicklungsseite, müssen eng kooperieren und die nötige Granularität aushandeln.

Interviewer:
Perfekt, vielen Dank für das Interview und deine Zeit!

ExpertIn:
Gern geschehen, ich bin gespannt auf das Endergebnis.

11.5.5 ExpertIn 5

Interviewer:
Dann haben wir ein paar demographische Fragen zu klären. Welches Kästchen darf ich hier bei deinem Alter ankreuzen?

ExpertIn:
Ja, 40 bis 50 bitte.

Interviewer:
Alles klar. Und was gibst du als Fachgebiet an?

ExpertIn:
IT und Test Management.

Interviewer:
Wie viele Jahre Erfahrung hast du in diesem Fachgebiet? Anschließend frage ich noch nach den Jahren insgesamt in der IT.

ExpertIn:
Nimm für Test Management 10 Jahre und für die gesamte IT 15 Jahre.

Interviewer:
Okay, notiert. Bei der technischen Ausbildung oder Weiterbildung? Hier ist Mehrfachauswahl möglich.

ExpertIn:
Universität, plus Weiterbildungen kannst du auch dazuschreiben.

Interviewer:
Perfekt. Dann kommen wir schon zur ersten Fachfrage: „In welcher Phase des Entwicklungsprozesses sehen Sie den Einsatz des Konzepts als sinnvoll an?“ (z.B. während der Anforderungserhebung, parallel zur Entwicklung, abschließend bei der Abnahme?)

ExpertIn:
Eigentlich über alle Phasen hinweg – mit leichter Abstufung. Ich glaube, der größte Nutzen kommt dann, wenn man schon ein laufendes Produkt hat, das man funktional erweitert und parallel dazu die Tests ausbaut. Aber man kann das sicher auch schon in einer ganz frühen Entwicklungsphase mit einbeziehen.

Interviewer:
Alles klar. Dann eine Skalenfrage: „Das Konzept, Fachexpert:innen ohne Programmierkenntnisse in den Entwicklungsprozess einzubinden, könnte den Entwicklungsprozess verbessern.“ Skala von 1 (stimme überhaupt nicht zu) bis 4 (stimme voll zu)?

ExpertIn:
Ich würde dem Ganzen eine 3 geben. Es kommt sehr auf das Projekt an, aber tendenziell glaube ich schon, dass es etwas verbessert.

Interviewer:
Wie stark könnte das Konzept deiner Meinung nach die Zusammenarbeit zwischen Fachexpert:innen und Entwickler:innen verbessern? Wieder 1 bis 4.

ExpertIn:
Auch eine 3. Wenn es gut in die Tool-Landschaft eingebunden ist, kann es Arbeitslast von der Entwicklung wegnehmen und asynchron bearbeitbar machen.

Interviewer:
Welche konkreten Vorteile siehst du in dem Konzept?

ExpertIn:

- **Asynchrone Bearbeitung**: Fachexpert:innen können Testfälle definieren, ohne dass man immer einen Entwickler:innen braucht.
- **Entlastung der Entwickler:innen**: Ein Teil der Testfall-Definition wandert direkt zum Fachexpert:innen.

- **Direkte Übersetzung fachlicher Use-Cases**: Man vermeidet Übersetzungsfehler, weil der Fachbereich selbst eingibt, was geprüft werden soll.

Interviewer:
Glaubst du, die Qualität oder die Abdeckung von Testfällen profitiert davon?

ExpertIn:
Ja, in der Breite sicher. Fachexpert:innen decken Use Cases ab, an die ein reiner Test Engineer nicht denken würde. Allerdings ersetzt das nicht die klassische Testautomatisierung in der Tiefe. Man braucht nach wie vor Tests, die Entwickler:innen oder QS-Spezialisten auf Code-Ebene schreiben.

Interviewer:
Welche Herausforderungen oder Bedenken fallen dir ein?

ExpertIn:

- **Kommunikation**: Auch wenn Fachexpert:innen Testfälle definieren, wird man viel Abstimmung brauchen.
- **Falsche Sicherheit**: Die Fachperson glaubt womöglich, alles sei nun getestet, obwohl die Implementierung vielleicht nicht 1:1 seiner Definition folgt.
- **Review-Prozess**: Man braucht eine Instanz, die kontrolliert, ob der Step und der Code wirklich übereinstimmen.

Interviewer:
Jetzt zu den spezifischen Vor- oder Nachteilen eines No-Code-Ansatzes im Vergleich zu „traditionellen“ Methoden, wo das Entwicklerteam die Tests schreibt?

ExpertIn:

- **Vorteil**: Man kann Personen ohne Codier-Kenntnisse direkt einbinden; das Fachwissen landet schneller in automatisierten Tests.
- **Nachteil**: Es braucht nach wie vor eine technische Kontrolle oder Koordination, damit das sauber umgesetzt wird.

Interviewer:
Und wie bewertest du das Ganze speziell für komplexe Projekte, z. B. mit Hardware- und Software-Komponenten?

ExpertIn:
Gerade bei Hardware-Themen ist die Einarbeitung für einen Test Engineer zeit- und ressourcenintensiv. Fachexpert:innen kennen die Details schon und können direkt sinnvolle Tests spezifizieren. Dadurch lässt sich insgesamt Zeit sparen und die Testabdeckung erhöhen.

Interviewer:
Würdest du in komplexen Projekten Fachexpert:innen mit diesem Konzept in den Entwicklungsprozess einbeziehen wollen?

ExpertIn:
Ja, auf jeden Fall. Das macht schon Sinn, wenn man sie entsprechend schult und unterstützt.

Interviewer:
Wie wichtig ist dabei die Integration in eine CI-Pipeline, also dass Fachexpert:innen vielleicht auch selbst Pipeline-Runs anstoßen können und Feedback erhalten?

ExpertIn:
Sehr wichtig. Das Feedback aus der CI, ob Tests grün oder rot sind, kann für Fachexpert:innen relevant sein – sofern man sie nicht überlädt. Aber grundsätzlich ist die Integration in die CI ein großer Faktor.

Interviewer:
Gut. Dann noch eine Skalenfrage: „Wie wahrscheinlich ist es, dass du eine Umsetzung dieses Konzepts in einem deiner nächsten Projekte verwendest?“ (1 = sehr unwahrscheinlich, 4 = sehr wahrscheinlich)

ExpertIn:
Ich nehme eine 3. Mit einer ausgereiften Plattform, die gut passt, würde ich es mir auf jeden Fall überlegen.

Interviewer:
Welche Bedingungen sind dafür nötig?

ExpertIn:

- **Einfache Plattform** mit guter Usability.
- **Integration in unser bestehendes TA-Framework** und vorhandene Prozesse.
- **Verfügbare Fachexpert:innen**, die das auch annehmen.

Interviewer:
Hast du noch Ideen oder Verbesserungen für das Konzept?

ExpertIn:

- **Status von Test-Schritten** anzeigen (Draft, Review, Freigegeben etc.).
- **Erweiterte Meta-Informationen**: Erwartetes Ergebnis, Kontext, mögliche Fehlerkategorien.
- **Bessere Lesbarkeit** der einzelnen Steps, etwa erwartete Resultate direkt sichtbar.

Interviewer:
Alles klar. Das war’s. Vielen Dank für deine Zeit und das Interview!

ExpertIn:
Gern geschehen. Danke ebenso!

Interviewer:
Ich stoppe jetzt die Aufzeichnung.