technische universität dortmund

IT in Produktion und Logistik

Konzept zur Einbindung von Natural Language Processing in Lifecycle-Management Applikationen

Masterarbeit zur Erlangung des Grades M. Sc

Vorgelegt von:	Carina Mariella Aretz
Matrikelnummer:	159223
Studiengang:	Maschinenbau

Ausgabedatum:	28.06.2024
Abgabedatum:	10.12.2024

Erstprüfer:	Dr.-Ing. Dipl.-Inf. Anne Antonia Scheidler
Zweitprüfer:	Dr. Thomas Hülser

Technische Universität Dortmund

Fakultät Maschinenbau

Fachgebiet IT in Produktion und Logistik

Abstract

Diese Arbeit befasst sich mit der Integration von Natural Language Processing (NLP) in Application Lifecycle Management (ALM) und untersucht die Automatisierung von Prozessen wie der Test Case-Generierung. Aufbauend auf dem CRISP-DM-Modell wird ein systematischer Ansatz entwickelt, der es ermöglicht, NLP effizient in bestehende ALM-Systeme einzubinden. Der Schwerpunkt der Arbeit liegt auf der Evaluation eines auf GPT-4o basierenden Tools, das Test Cases aus Anforderungen generiert, und dem Vergleich mit einem spezialisierten LLM-basierten Tool.
Die Ergebnisse zeigen, dass NLP erhebliches Potenzial besitzt, die Effizienz und Qualität in ALM-Prozessen zu steigern. Insbesondere bei klar definierten Anforderungen können präzise und konsistente Test Cases generiert werden. Jedoch wurde auch deutlich, dass unklare oder unvollständige Anforderungen, mathematische Berechnungen und Edge Cases Herausforderungen für die Modelle darstellen. Der Vergleich der Tools verdeutlicht, dass allgemeine LLMs häufig konsistentere und präzisere Ergebnisse liefern, während spezialisierte Modelle in domänenspezifischen Anwendungsfällen Vorteile aufweisen können.
Diese Arbeit liefert Erkenntnisse für die zukünftige Integration von NLP in ALM und identifiziert Optimierungspotenziale, etwa in der Berücksichtigung komplexer Anforderungen und der Verbesserung der Konsistenz von Test Cases. Die vorgestellten Ansätze und Erkenntnisse bilden eine Grundlage für weiterführende Forschung und die praktische Anwendung von NLP im Softwareentwicklungsprozess.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ALM	Application Lifecycle Management
API	Anwendungsprogrammschnittstelle
CSS	Cascading Style Sheets
CSV	Comma-seperated values
DL	Deep Learning
EP	Echt positiv
FN	Falsch negativ
FP	Falsch positiv
HTML	HyperText Markup Language
JI	Jaccard-Index
JML	Java Modeling Language
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
LM	Lifecycle Management
ML	Maschinelles Lernen
NER	Named Entity Recognition
NLTK	Natural Language Toolkit
OEM	Original Equipment Manufacturers
PDM	Produktionsdatenmanagement
PLM	Product Lifecycle Management
RAD	Rapid Application Development
re	Regular Expressions
RUP	Rational Unified Process
SDLC	Software Development Lifecycle
SE	Systems Engineering
SGML	Standard Generalized Markup Language
SRS	Software Requirements Specification
TF-IDF	Term Frequency-Inverse Document Frequency
UML	Unified Modeling Language
XML	Extensible Markup Language
XP	Xtreme Programming
YAML	Yet Another Markup Language

1 Einleitung

Die Verwendung von Natural Language Processing (NLP) genießt seit Veröffentlichung von ChatGPT im Jahr 2022 besonderes Interesse der breiten Bevölkerung. NLP beschreibt computergestützte Techniken zur maschinellen Erkennung und Verarbeitung natürlicher Sprache. Ziel ist es, die direkte Kommunikation zwischen Mensch und Computer zu ermöglichen und die Analyse der zunehmenden Textmengen zu automatisieren.

Im Kontext des Lifecycle Managements (LM), das den gesamten Lebenszyklus eines Produkts oder einer Applikation von der ersten Idee bis zur Entsorgung umfasst, bietet NLP innovative Möglichkeiten, unter anderem zur Effizienzsteigerung und Qualitätsverbesserung (Guo et al. 2024; Bernard et al. 2017; Wang et al. 2021; Bouras et al. 2016). Durch die Automatisierung von Routineaufgaben wie Kundenfeedback-Analyse und Dokumentationsverarbeitung können Zeit und Kosten gespart werden. Zudem ermöglicht NLP eine präzise und konsistente Verarbeitung großer Datenmengen, wodurch die Informationsqualität erheblich verbessert werden kann (Min et al. 2021).

Die Forschung zu NLP und LM ist auf die Entwicklung von Algorithmen und Modellen konzentriert, die große Textmengen auswerten und relevante Informationen extrahieren können. Fortschritte in den Bereichen maschinelles Lernen und Deep Learning treiben diese Entwicklungen voran und eröffnen neue Anwendungsmöglichkeiten, wie die automatische Analyse von Markttrends und wissenschaftlichen Veröffentlichungen (Bharadiya 2023).

Insgesamt trägt die Integration von NLP in LM-Applikationen zur Effizienzsteigerung, Innovationsförderung und Verbesserung der bereichsübergreifenden Zusammenarbeit bei (Just 2024). Angesichts der kontinuierlich wachsenden Datenmengen und der zunehmenden Produktkomplexität wird die Bedeutung dieses Themas in der Zukunft weiter zunehmen (Ogundipe et al. 2024).

Die Fortschritte im Bereich der künstlichen Intelligenz (KI) und des maschinellen Lernens, insbesondere durch Deep Learning (DL), haben die Leistungsfähigkeit von NLP-Systemen erheblich gesteigert. Die aktuellen Modelle sind in der Lage, natürliche Sprache genauer und umfassender zu verstehen und zu verarbeiten als je zuvor (Khurana et al. 2023). Diese Entwicklungen ermöglichen es, NLP in immer komplexeren und datenintensiveren Bereichen wie dem Application Lifecycle Management (ALM) einzusetzen. Zudem wächst die Menge an verfügbaren Daten exponentiell, wodurch sich die Notwendigkeit und das Potenzial für automatisierte Analysetools weiter verstärkt (Bouras et al. 2016).

Trotz der Fortschritte in der NLP-Forschung gibt es noch erhebliche Lücken in der Anwendung dieser Technologien auf spezifische Bereiche wie ALM. Obwohl zahlreiche Studien zur allgemeinen Verbesserung von NLP-Algorithmen existieren, ist die Integration von NLP in spezifische Prozesse und Workflows des ALM oder PLM in der einschlägigen Literatur kaum thematisiert. Dies betrifft insbesondere die Anpassung und Optimierung von NLP-Techniken zur effizienten Verarbeitung und Analyse der umfangreichen und oft heterogenen Datenmengen, die im LM anfallen. Diese Forschungslücke bietet ein großes Potenzial für innovative Ansätze und praktische Anwendungen.

Externe Faktoren wie die zunehmende Digitalisierung und Globalisierung der Wirtschaft treiben die Relevanz des Themas zusätzlich voran. Unternehmen stehen unter ständigem Druck, ihre Entwicklungsprozesse zu optimieren, um wettbewerbsfähig zu bleiben. Die Fähigkeit, schnell auf Marktveränderungen und Kundenfeedback zu reagieren, wird immer entscheidender. NLP kann hier als Schlüsseltechnologie dienen, indem es die Verarbeitung und Analyse großer Mengen an Textdaten automatisiert und wertvolle Einblicke liefert.

Auf Basis der dargelegten Überlegungen und Erkenntnisse kristallisieren sich folgende Forschungsfragen heraus, die im Zentrum der vorliegenden Arbeit stehen werden:

F1: „Wie kann Natural Language Processing wertschöpfend für Application Lifecycle Management Anwendungen genutzt werden?“

F2: „Wie kann die Integration von Natural Language Processing im Application Lifecycle Management durchgeführt werden?“

Um diese Forschungsfragen zu beantworten, wird diese Arbeit verschiedene NLP-Modelle und -Techniken untersuchen, um ihre Stärken, Schwächen und Anwendbarkeit im ALM-Kontext zu bewerten. Das Konzept des Natural Language Processings wird vorgestellt, sowie verschiedene Definitionen kurz verglichen und erklärt. Es werden verschiedene Möglichkeiten zur Umsetzung beleuchtet, sowie Open Source Angebote und Angebote von Dienstleistern wie OpenAI aufgezeigt, um die Potentiale und Grenzen der Technologien im Kontext spezifischer Anwendungsfälle zu bewerten. Zur Ermittlung und Bewertung der Anwendungsmöglichkeiten wird eine systematische Literaturrecherche durchgeführt.

Im Bereich Lifecycle Management wird speziell das ALM untersucht. Dazu gibt es verschiedene Definitionen, die kurz besprochen werden. Dann wird anhand des Vorgehensmodells CRISP-DM ein Konzept zur beispielhaften Implementierung von NLP in den Kontext von ALM-Systemen entwickelt und umgesetzt. Dieses Modell soll dabei helfen, die notwendigen Schritte von der Datenerfassung über die Modellierung und Optimierung bis hin zur Auswertung und Implementierung zu durchlaufen. Die Arbeit wird auch ein konkretes Beispiel für die Umsetzung dieses Modells präsentieren. Diese beispielhafte Implementierung wird im Anschluss dann evaluiert und auch mit einer Benchmark verglichen. Dafür wird das NLP-Tool von einer Firma herangezogen, die sich auf die Entwicklung von KI-Lösungen für das Application Lifecycle Management spezialisiert hat. Die Gründer des Unternehmens haben zuvor ein ALM Tool namens Codebeamer entwickelt, welches in der Automobilbranche Anwendung findet. Das Tool wird in der Evaluierung als Benchmark herangezogen, da es ein aktuelles State-Of-The-Art Tool im Bereich KI in ALM ist.

Im Zuge dieser Arbeit wird auch darauf eingegangen, wie zukünftige Entwicklungen in der KI- und NLP-Forschung die Fähigkeiten und Anwendungsmöglichkeiten von NLP im ALM weiter verbessern werden. Schließlich wird ein Ausblick auf zukünftige Forschungsmöglichkeiten in diesem Bereich gegeben. Dabei wird sie potenzielle neue Anwendungsfelder für NLP im ALM, die Notwendigkeit weiterer empirischer Studien und die Möglichkeit tiefergehender Untersuchungen bestimmter Aspekte des NLP-ALM-Themas identifizieren. In dieser Arbeit wird auch diskutiert, wie die Forschungsergebnisse genutzt werden könnten, um praxisorientierte Leitfäden, Tools und Best Practices für die Integration von NLP in das ALM zu entwickeln. Im nächsten Schritt werden die technologischen Grundlagen erläutert, die zum Verständnis dieses Themenspektrums benötigt werden, und ein kurzer Überblick über den Stand der Technik gegeben.

2 Technologische Grundlagen und Stand der Technik

In diesem Kapitel werden die zur Beantwortung der Forschungsfragen benötigten Grundlagen zur Nutzung und Integration von Natural Language Processing im Application Lifecycle Management aufgearbeitet und erläutert. Es wird auf die relevanten Konstrukte, Modelle und Definitionen eingegangen, die benötigt werden, um die weiteren Inhalte verständlich zu machen. Zunächst wird das ALM im Hinblick auf seine Definition, Zielsetzung und Anwendungsbereiche analysiert, bevor im Anschluss das Konzept von Natural Language Processing aufgearbeitet. Anschließend werden die Anwendungsmöglichkeiten von NLP im ALM mit Hilfe einer systematischen Literaturrecherche ermittelt.

2.1 Vorgehensmodelle in der Softwareentwicklung

In diesem Kapitel wird ein zentrales Vorgehensmodelle in der Softwareentwicklung vorgestellt: das CRISP-DM (Cross-Industry Standard Process for Data Mining) Modell und wurde erstmals von Chapman et al. im Jahr 2000 vorgestellt. Neben diesen traditionellen Modellen wie dem V-Modell, gewinnen agile Methoden in der Softwareentwicklung zunehmend an Bedeutung. Agile Ansätze, wie Scrum, Kanban oder Extreme Programming (XP), zeichnen sich durch ihre iterative Arbeitsweise, hohe Flexibilität und die kontinuierliche Einbindung von Stakeholdern aus. Auch wenn auf diese Methoden an dieser Stelle nicht weiter eingegangen wird, ist es wichtig, ihre wachsende Relevanz in modernen Entwicklungsprojekten zu erwähnen.

CRISP-DM

Das CRISP-DM Modell ist ein weit verbreitetes Prozessmodell für Data Mining-Projekte. Es bietet eine strukturierte Vorgehensweise, die aus sechs Phasen besteht, die iterativ und flexibel durchlaufen werden können, um die Anforderungen eines Projekts optimal zu erfüllen.

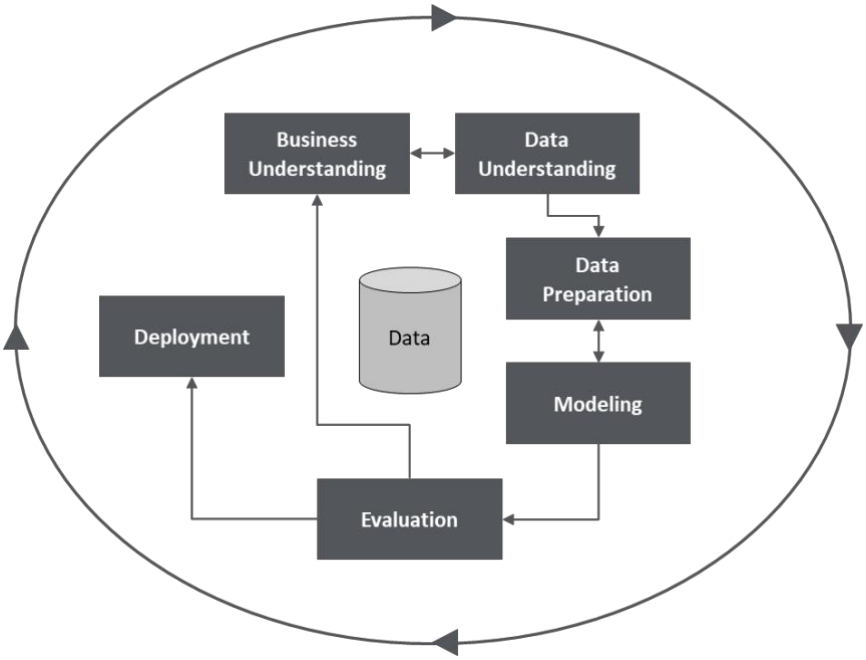


Abbildung 2-1 CRISP-DM Modell (Chapman et al. 2000)

In der ersten Phase, Business Understanding, liegt der Fokus darauf, die Geschäftsziele und -anforderungen des Projekts zu verstehen. Es geht darum, das Problem klar zu definieren und die Erwartungen an das Projekt zu klären. Diese Phase bildet die Grundlage für alle folgenden Schritte, da sichergestellt ist, dass das Data Mining-Projekt auf die Bedürfnisse des Unternehmens ausgerichtet ist.

Anschließend folgt die Phase Data Understanding, in der die verfügbaren Daten gesammelt und explorativ analysiert werden. Ziel ist es, ein tiefes Verständnis für die Daten zu entwickeln, um ihre Eignung für das Projekt zu beurteilen und mögliche Probleme frühzeitig zu erkennen.

In der Data Preparation-Phase werden die Daten für die Modellierung aufbereitet. Dies umfasst Aufgaben wie das Bereinigen der Daten, die Auswahl relevanter Variablen und das Erstellen von Datensätzen, die für die Analyse geeignet sind. Diese Phase ist oft sehr zeitaufwendig, da die Qualität der Daten maßgeblich den Erfolg des Projekts beeinflusst.

Die nächste Phase, Modeling, beinhaltet die Anwendung von verschiedenen Data-Mining-Methoden auf die vorbereiteten Daten, um Muster oder Modelle zu erzeugen. Es können verschiedene Techniken und Algorithmen ausprobiert werden, um die bestmöglichen Ergebnisse zu erzielen. Dabei ist es oft notwendig, zur vorherigen Phase zurückzukehren und die Datenvorbereitung anzupassen, um bessere Modelle zu erhalten.

Nach der Modellierung folgt die Evaluation-Phase, in der die erstellten Modelle gründlich getestet und bewertet werden. Ziel ist es, zu überprüfen, ob die Modelle die ursprünglichen Geschäftsziele erfüllen und ob sie robust und verlässlich sind. Diese Phase ist entscheidend, um sicherzustellen, dass die Ergebnisse des Data Mining-Prozesses im praktischen Einsatz nützlich sind.

Die letzte Phase des CRISP-DM Modells ist die Deployment-Phase. Hier werden die Ergebnisse des Projekts in die Praxis umgesetzt. Dies kann in Form einer einfachen Präsentation der Ergebnisse erfolgen oder als Implementierung der Modelle in das operative Geschäft des Unternehmens. Je nach Anwendungsfall kann dies auch eine umfassende Systemintegration bedeuten.

CRISP-DM ist ein flexibles und iteratives Modell, das es ermöglicht, jederzeit zwischen den Phasen zu wechseln, um das Projekt kontinuierlich zu verbessern. Es wird durch den kreisförmigen Fluss im Modell verdeutlicht, der zeigt, dass Data Mining-Projekte selten linear verlaufen und ständiger Anpassung bedürfen.

2.2 Auszeichnungssprachen

Auszeichnungssprachen, im Englischen als "markup languages" bezeichnet, sind strukturierte Sprachen, die zur Formatierung und Strukturierung von Daten verwendet werden. Ihr Hauptzweck besteht darin, Inhalte durch sogenannte "Tags" (Markierungen) zu kennzeichnen, die sowohl den Aufbau als auch die Darstellung der Daten beschreiben können. Diese Tags sind meist in spitzen Klammern < > eingefasst und definieren die Bedeutung oder den Zweck bestimmter Daten. Die grundlegende Idee hinter Auszeichnungssprachen ist die Trennung von Inhalt, Struktur und Darstellung, wodurch Daten sowohl maschinenlesbar als auch für Menschen verständlich gemacht werden können.

Es gibt eine Vielzahl von Auszeichnungssprachen, die für unterschiedliche Anwendungszwecke entwickelt wurden. Zu den bekanntesten zählen:

Tabelle 2-1 Überblick über Auszeichnungssprachen

Name	Beschreibung
XML	XML (Extensible Markup Language) ist eine flexible und erweiterbare Auszeichnungssprache, die speziell für den Datenaustausch zwischen Systemen konzipiert wurde. Sie dient dazu, Daten unabhängig von der Plattform oder Programmiersprache zu strukturieren und zu speichern (World Wide Web Consortium 2006).
HTML	HTML (HyperText Markup Language) ist die Grundlage für die Darstellung von Webseiten. Es ermöglicht die Strukturierung von Inhalten wie Text, Bildern und Links und wird in Kombination mit CSS

	(Cascading Style Sheets) und JavaScript verwendet, um interaktive und ansprechende Webanwendungen zu erstellen (World Wide Web Consortium 1999).
Markdown	Eine leichtgewichtige Auszeichnungssprache, die häufig für Dokumentationen, Blogs oder Readme-Dateien in der Softwareentwicklung genutzt wird. Sie ermöglicht die einfache Formatierung von Texten ohne komplexe Syntax (Gruber 2012).
LaTeX	Eine weit verbreitete Sprache zur Erstellung von wissenschaftlichen Texten, insbesondere für mathematische oder technische Dokumentationen (LaTeX 2024).
YAML und JSON	Obwohl sie streng genommen keine klassischen Auszeichnungssprachen sind, dienen YAML (Yet Another Markup Language) und JSON (JavaScript Object Notation) ähnlichen Zwecken, nämlich der strukturierten Datenrepräsentation in einer leicht lesbaren und maschinenverarbeitbaren Form (Evans et al. 2009; W3Schools 2024b).

Unter den genannten Auszeichnungssprachen nimmt XML aufgrund seiner Flexibilität und Plattformunabhängigkeit eine besondere Rolle ein. XML wurde 1998 vom World Wide Web Consortium (W3C) entwickelt und ist eine vereinfachte Version der Standard Generalized Markup Language (SGML). XML ist nicht darauf ausgelegt, Inhalte darzustellen, sondern vielmehr, Daten in einer strukturierten, hierarchischen Weise zu organisieren. XML-Dokumente basieren auf einer klar definierten Baumstruktur, die es ermöglicht, Daten hierarchisch und logisch zu organisieren. Ein typisches XML-Dokument beginnt mit einer Deklaration, die die Version und die Zeichenkodierung festlegt, wie zum Beispiel <?xml version="1.0" encoding="UTF-8"?>. Anschließend folgt der eigentliche Inhalt, der aus Elementen, Attributen und Text besteht. Elemente bilden die grundlegenden Bausteine und werden durch ein öffnendes Tag (z. B. <element>) eingeleitet und durch ein schließendes Tag (z. B. </element>) beendet. Innerhalb der Tags können Textinhalte oder verschachtelte Unterelemente enthalten sein, wodurch eine Baumstruktur entsteht. Attribute bieten eine Möglichkeit, zusätzliche Informationen zu den Elementen hinzuzufügen. Diese werden innerhalb des öffnenden Tags definiert, beispielsweise als attribute="value". Die klare Hierarchie und Verschachtelung machen XML zu einer idealen Sprache für die Abbildung komplexer Datenstrukturen (World Wide Web Consortium 2006).

Obwohl XML keine vordefinierten Tags besitzt, gibt es feste Syntaxregeln, die eingehalten werden müssen, um als "well-formed" (wohlgeformt) zu gelten. Jedes Tag muss korrekt geschlossen werden, und die Verschachtelung von Elementen darf sich nicht überschneiden. Attribute müssen in Anführungszeichen gesetzt werden, und Sonderzeichen, die als Teil der XML-Syntax interpretiert werden könnten, wie < oder &, müssen entsprechend kodiert werden. XML unterstützt außerdem die Verwendung von Namespaces, um Namenskollisionen bei der Integration verschiedener XML-Vokabulare zu vermeiden. Dies geschieht durch die Angabe eines Präfixes in Verbindung mit einem Namespace, z. B. <ns:element xmlns:ns=" http://www.w3.org/2001/XMLSchema">. Für Textinhalte, die Sonderzeichen enthalten, bietet XML die Möglichkeit, diese zu schützen, um eine fehlerhafte Interpretation zu vermeiden. XML findet in zahlreichen Anwendungsbereichen Verwendung. Ein zentraler Einsatzbereich ist der plattformübergreifende Datenaustausch, insbesondere in Webservices wie REST oder SOAP, bei denen strukturierte Daten zwischen Systemen ausgetauscht werden. Darüber hinaus dient XML häufig als Format für Konfigurationsdateien in Softwareanwendungen, beispielsweise in pom.xml für Maven-Projekte oder AndroidManifest.xml in Android-Apps. Auch in der Dokumentenspeicherung spielt XML eine wichtige Rolle, insbesondere in standardisierten Dateiformaten wie DOCX oder ODT, die auf XML-basierenden Strukturen aufbauen. Die Flexibilität von XML ermöglicht zudem die Anpassung an domänenspezifische Anforderungen

und macht es zu einer weit verbreiteten Technologie in verschiedensten Branchen (W3Schools 2024c).

XML bietet zahlreiche Vorteile, die es zu einer beliebten Wahl für die Datenstrukturierung machen. Besonders hervorzuheben ist die Plattformunabhängigkeit, die sicherstellt, dass XML-Daten in nahezu jedem System verarbeitet werden können. Die Möglichkeit, eigene Tags zu definieren, verleiht XML eine hohe Flexibilität, während die Unterstützung durch eine Vielzahl von Tools und Standards wie XSLT, XPath und XQuery die Funktionalität erweitert. Gleichzeitig hat XML jedoch auch einige Schwächen. Der Speicheraufwand ist oft höher als bei Alternativen wie JSON, da XML-Tags sowohl öffnend als auch schließend angegeben werden müssen. Dies kann zu einer erhöhten Dateigröße führen, insbesondere bei großen Datenmengen. Zudem erfordert die Erstellung und Verarbeitung komplexer XML-Strukturen oft spezielle Tools oder Bibliotheken, wodurch die Komplexität im Vergleich zu leichtergewichtigen (Lightweight) Formaten erhöht wird. Trotz dieser Herausforderungen bleibt XML aufgrund seiner Vielseitigkeit und Standardisierung ein unverzichtbares Werkzeug in der Datenverarbeitung (World Wide Web Consortium 2010).

Eine weitere herausstechende Auszeichnungssprache ist HTML ist die Standard-Auszeichnungssprache zur Strukturierung und Darstellung von Inhalten im World Wide Web. Es wurde entwickelt, um Texte, Bilder, Links und Multimedia-Inhalte in einem Browser ansprechend und funktional darzustellen. Der grundlegende Aufbau eines HTML-Dokuments beginnt mit einer Deklaration <!DOCTYPE html> und besteht aus einer hierarchischen Struktur, die aus Elementen mit öffnenden und schließenden Tags wie <html>, <head> und <body> besteht. Im <head>-Bereich werden Metadaten wie Titel, Stylesheets oder Skripte definiert, während der <body>-Bereich die eigentlichen Inhalte der Webseite enthält. Die Nutzungsmöglichkeiten von HTML sind vielfältig: Es ermöglicht die Erstellung von Webseiten mit Texten, Bildern, Tabellen, Formularen und eingebetteten Medien. In Kombination mit CSS und JavaScript lassen sich das Design und die Funktionalität weiter ausbauen, wodurch interaktive und dynamische Webanwendungen entstehen. HTML ist zentral für die Strukturierung von Inhalten, während CSS und JavaScript die Darstellung und das Verhalten ergänzen (W3Schools 2024a).

Im Vergleich zu XML unterscheiden sich die beiden Sprachen in Zweck und Flexibilität. Während HTML eine festgelegte Struktur und einen klar definierten Tag-Satz besitzt, der speziell für die Darstellung von Webinhalten optimiert ist, ist XML wesentlich flexibler und dient primär der Strukturierung und Speicherung von Daten. XML erlaubt Benutzern die Definition eigener Tags, wodurch es für den Datenaustausch und die Speicherung komplexer Datenmodelle geeignet ist. HTML hingegen ist weniger flexibel, aber einfacher zu nutzen und speziell auf die Anforderungen des Webs abgestimmt. Beide Sprachen ergänzen sich jedoch in vielen Anwendungen, insbesondere wenn XML-Daten in HTML-Seiten eingebettet und dargestellt werden sollen. Beide Formate sind essentielle Bausteine in der digitalen Vermittlung von Daten. XML bildet die Grundlage vieler Datenformate und ist in ALM-Systemen weit verbreitet, um strukturierte Informationen zu speichern und auszutauschen. HTML ist ein Schlüsseltechnologie für die Entwicklung von Webapplikationen, die häufig als Benutzeroberfläche für ALM-Systeme dienen. Ein grundlegendes Verständnis beider Technologien ist daher wichtig, um die bestehende Funktionalität von ALM-Systemen nachvollziehen zu können und deren Weiterentwicklung gezielt zu unterstützen.

2.3 Application Lifecycle Management

In diesem Abschnitt wird der Hintergrund des ALM erörtert. Zunächst erfolgt eine Einführung in das grundlegende Konzept des ALM. Darauf aufbauend werden zentrale Themenbereiche wie Anforderungsmanagement und Rückverfolgbarkeit, Konfigurationsmanagement sowie Toolintegration thematisiert. Abschließend wird eine Betrachtung von ALM im Kontext der eingesetzten Tools vorgenommen.

2.3.1 Das Konzept des Application Lifecycle Management

Anfang des 21. Jahrhunderts wurden die ersten Konzepte des Application Lifecycle Management vorgestellt, das die Koordinierung von Aktivitäten und die Verwaltung von Artefakten (z. B. Anforderungen, Quellcode, Test Cases) während des Lebenszyklus von Softwareprodukten bezeichnet. ALM wird manchmal mit dem Software Development Life

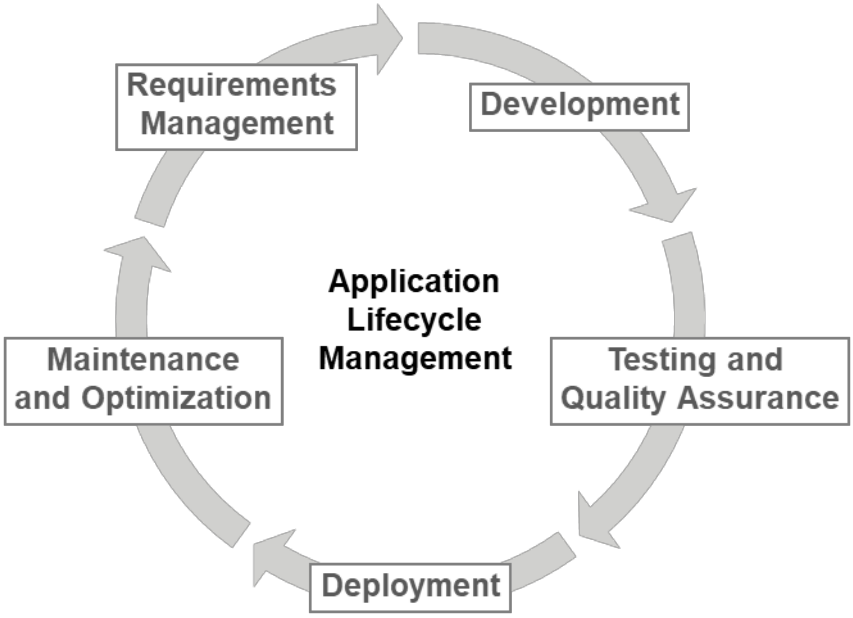


Abbildung 2-2 Bestandteile des Application Lifecycle Managements (Eigene Darstellung in Anlehnung an Amazon Web Services (2024) und PTC (2024))

Cycle (SDLC) verwechselt, da sich beide einen Prozess der Softwareentwicklung beschreiben (Chappell 2008; Goth 2009).

Der Hauptunterschied besteht darin, dass sich der SDLC in erster Linie auf die Entwicklungsphase konzentriert, während sich ALM mit dem gesamten Lebenszyklus der Anwendung befasst. Dieser reicht vom Konzept über die Wartung bis hin zur Stilllegung, und umfasst auch die Zeit nach der Entwicklungsphase. David Chapell (2008, S. 2f.) prägte den Begriff in den Anfängen maßgeblich und teilte den Begriff in drei Aspekte auf: Verwaltung, Entwicklung und Betrieb. Mittlerweile hat sich das Konzept weiterentwickelt. Die meisten Anbieter wie AWS (2024), IBM (2024), Siemens (2024), SAP (2024), Salesforce (2024), Microsoft (2024), Atlassian (2024) und PTC (2024) teilen das ALM in fünf Phasen ein: Anforderungsmanagement, Entwicklung, Prüfung und Qualitätssicherung, Bereitstellung sowie Instandhaltung und Optimierung. ALM kann als der strategische Prozess der Verwaltung eines Software- oder Produktlebenszyklus von der ersten Idee über Design, Entwicklung, Tests und Bereitstellung bis hin zum Ende der Lebensdauer verstanden werden.

2.3.2 Abgrenzung zu verwandten Begriffen

Um ALM im Kontext einzuordnen, werden in diesem Abschnitt verwandte Begriffe erklärt und in den Zusammenhang mit ALM gesetzt. Mit Hilfe von ALM können Prozesse, Tools und Ressourcen in den verschiedenen Phasen des Softwareentwicklungszyklus verwaltet und koordiniert werden, einschließlich Anforderungserfassung, Design, Codierung, Tests, Freigabe und Wartung. ALM integriert Aufgaben der Entwicklung, Zusammenarbeit, Kommunikation und des Wissensmanagements und zentralisiert die Verwaltung von Benutzern, Projekten und Prozessen. Es geht über den typischen Software Development Lifecycle (SDLC) hinaus, indem es auch Phasen nach der Entwicklung wie Bereitstellung, Betrieb und Optimierung einbezieht.

Im Gegensatz dazu konzentriert sich das PLM auf die effektive Verwaltung der Produkte eines Unternehmens innerhalb ihres gesamten Lebenszyklus. Dabei werden alle Aktivitäten und

Datenflüsse während des physischen Produktentwicklungsprozesses sowie während der Wartung und des Supports verfolgt und verwaltet. PLM integriert alle Elemente - Menschen, Prozesse, Geschäftssysteme und Informationen -, die an der Produktentwicklung beteiligt sind und deren Lebenszyklus entlang der Wertschöpfungskette unterstützen. Während sich PLM in erster Linie mit physischen Produkten befasst, enthält ALM softwarespezifische Erweiterungen, um der dynamischen Natur von Softwareprodukten Rechnung zu tragen, z. B. Aktivitäten nach der Freigabe und wesentliche Funktionsänderungen. Deuter und Rizzo (2016, S.409) heben ebenfalls hervor, dass PLM aufgrund seiner engen Verbindung zum Hardware-bezogenen Lebenszyklus an seine Grenzen stößt, wenn die Software nicht berücksichtigt wird. Aus diesem Grund wurde ALM eingeführt, um die Software als zentrale Komponente in die Lebenszyklusentwicklung und -verwaltung einzubinden.

Der SDLC ist ein systematischer Prozess, der darauf abzielt, Software mit optimaler Qualität, minimalen Kosten und innerhalb eines kurzen Zeitrahmens zu liefern. Er umfasst spezifische Schritte und genau definierte Phasen, um effizient gründlich getestete und einsatzbereite Software zu produzieren. Gängige SDLC-Modelle sind Wasserfall, Spiral, inkrementell oder iterativ, Rational Unified Process (RUP), Rapid Application Development (RAD), V-Modell, Agile, Synchronisieren und Stabilisieren und Rapid Prototyping. Im Gegensatz zu ALM konzentriert sich SDLC nur auf die Entwicklungs- und anfänglichen Einsatzphasen von Software und deckt die Einsatz-, Betriebs- und Wartungsphasen nicht umfassend ab.

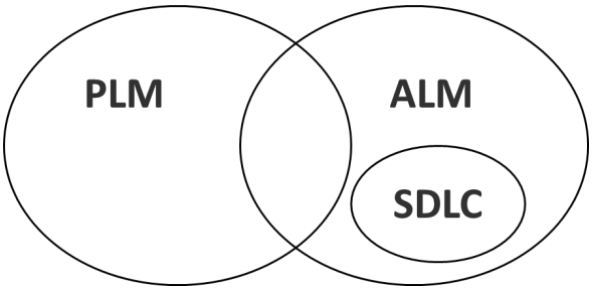


Abbildung 2-3 Beziehung zwischen PLM, ALM und SDLC (Jakab (2024, S.18))

Zusammenfassend lässt sich festhalten, dass ALM den gesamten Lebenszyklus einer Softwareanwendung abdeckt, indem es sowohl Entwicklungs- als auch Betriebsaspekte integriert, um eine kontinuierliche Verwaltung und Optimierung zu gewährleisten. Im Gegensatz dazu konzentriert sich PLM auf den Lebenszyklus physischer Produkte, wobei alle Phasen von der Entwicklung über die Produktion bis hin zur Wartung berücksichtigt werden. SDLC beschränkt sich hingegen auf die Entwicklungs- und anfänglichen Implementierungsphasen von Softwareprojekten. Während ALM eine breitere Palette von Artefakten verwaltet, einschließlich solcher, die nach der Bereitstellung einer Software erforderlich sind, fokussiert sich PLM auf Artefakte, die mit der physischen Produktentwicklung verbunden sind. SDLC konzentriert sich wiederum auf die in den Softwareentwicklungsphasen benötigten Artefakte. Insgesamt bietet ALM ein integriertes Ökosystem von Tools und Prozessen zur Verwaltung von Software über ihren gesamten Lebenszyklus, während PLM Menschen, Prozesse und Systeme vereint, die an der physischen Produktentwicklung beteiligt sind. SDLC stellt spezifische Methoden zur Verwaltung des Softwareentwicklungsprozesses bereit.

2.3.3 Application Lifecycle Management Tools

Goth (2009, S.88ff.) hob bereits früh hervor, dass die Nachfrage nach ALM-Tools für die agile Entwicklung rasch zunimmt. Wie in Kapitel 2.1.1 erläutert, hat die Bedeutung von ALM für die Industrie weiter zugenommen, wodurch verschiedene Tools zum Management des Application Lifecycles auf den Markt kamen. In diesem Unterkapitel wird ein kurzer Überblick über die bestehende ALM-Tool-Landschaft gegeben.

Die Merkmale von ALM-Werkzeugen wurden in einem wissenschaftlichen Artikel von Simone et al. (2018) gesammelt und zusammengefasst. In Abbildung 2-4 werden verschiedene IT Lösungen gezeigt, die ein ALM-Werkzeug auf dem Markt enthalten kann.

ALM IT Lösungen
Agiles Projektmanagement
Release Management
Anforderungsmanagement
Dokumentenmanagement
Integration von Software-Entwicklungswerkzeugen
Quellcode Management (Versionskontrolle)
Integration von Software-Erstellungsprozessen
Test Management
Workflow-Unterstützung und Task-/Ticket-Management
Fehler- und Problemverfolgung
Software-Konfigurationsmanagement
Verwaltung von Standardbibliotheken

Abbildung 2-4 Application Lifecyclemanagement IT Lösungen (Deuter et al. (2019, S.126))

Besonders sprachintensive Bereiche der ALM-IT-Lösungen sind das Anforderungsmanagement, das Test Management, die Fehler- und Problemverfolgung sowie das Dokumentenmanagement. Im Anforderungsmanagement werden Anforderungen in natürlicher Sprache erfasst, analysiert und verwaltet, wobei eine präzise und eindeutige Formulierung entscheidend ist. Das Test Management steht in engem Zusammenhang mit dem Anforderungsmanagement, da für jede Produktanforderung ein entsprechender Test Case gestaltet werden muss, um die Erfüllung der Anforderung zu überprüfen. Die beiden Begriffe können auch unter dem Begriff des Requirements Engineering zusammengefasst werden (Rupp und Pohl 2022). Auch die Fehler- und Problemverfolgung erfordert eine detaillierte sprachliche Beschreibung von Fehlern und deren Ursachen, um eine effiziente Behebung zu ermöglichen. Das Dokumentenmanagement schließlich sorgt für die Erstellung, Organisation und Bereitstellung aller textbasierten projektrelevanten Inhalte. Diese Bereiche verdeutlichen, wie stark Sprache und Text in ALM-Prozessen integriert sind.

Viele ALM Tools auf dem Markt umfassen dabei mindestens 2 dieser Lösungen. Bekannte Beispiele wie Jira von Atlassian (2024) und Codebeamer von PTC (2024) enthalten alle Lösungen in einem. Moreira (2013) vertritt den Standpunkt, dass es keine umfassende ALM-Lösung gibt, die allen Anforderungen gerecht wird, da der Umfang und die Komplexität eines vollständigen ALM sowie die immer kompliziertere und vielfältigere Art der Software-Entwicklung zu groß sind. Je höher jedoch der Integrationsgrad eines Tool-Frameworks ist, desto mehr kann sich ein agiles Team auf die Schaffung von Kundennutzen konzentrieren. Für die richtige Werkzeugauswahl haben Klespitz et al. (2016) eine Empfehlung für Unternehmen erstellt, um die richtigen ALM-Lösungen für ihren Zweck auszuwählen. In Abbildung 2-5 ist eine Übersicht von Anbietern von ALM-Tools, sowie eine Einordnung dieser von Hastie (2015) zu sehen. Diese Übersicht ist inzwischen aufgrund ihres Alters nicht mehr aktuell, spiegelt jedoch nach wie vor die wesentlichen Marktverhältnisse treffend wider.

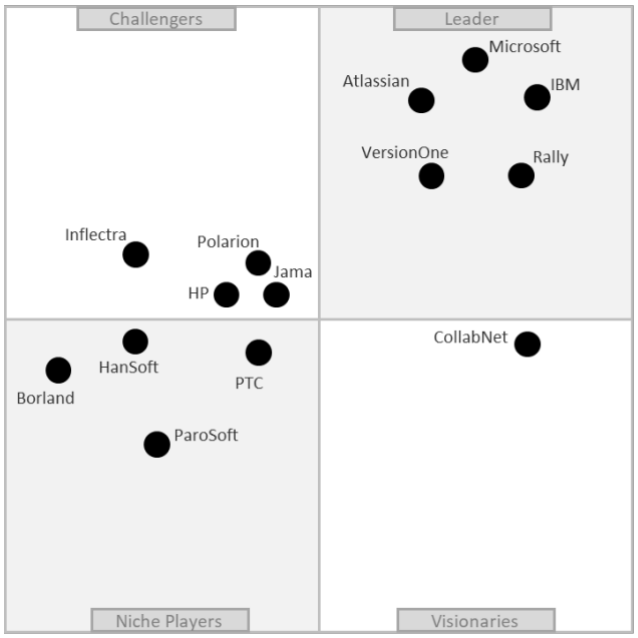


Abbildung 2-5 Anbieter von ALM-Tools und ihre Fähigkeiten (Hastie (2015, S.3))

Auch wenn ALM-Anbieter, darunter prominente Unternehmen aus dem Bereich der SW-Tools wie Microsoft, IBM und HP einen hohen Fokus auf die größtmögliche Wertschöpfung ihrer Produkte legen, gibt es laut Regan et al. (2015, S.606f.) einige allgemeine inhärente Schwächen:

- Die Traceability die Nachvollziehbarkeit ist primär auf das mitgelieferte ALM System beschränkt. Es gibt Anwendungsprogrammschnittstellen (APIs) für den Zugriff auf interne Daten, aber bis zur Einführung von OSLC gab es keine definierte offene Methode zum Austausch dieser Daten.
- Es können Rückverfolgbarkeitsberichte erstellt werden, die wertvolle Informationen liefern. Diese Berichte sind jedoch statisch und spiegeln nicht den dynamischen Charakter von Anforderungen und erkannten Problemen wider, die auch von Quellen außerhalb des ALM-Systems stammen können.
- Die Komplexität der Widgets, einschließlich Schaltflächen, Textfelder, Registerkarten und Links, die für den Zugriff auf und die Bearbeitung von Ressourceneigenschaften zur Verfügung stehen, kann Prüfer und Benutzer leicht verwirren.
- Prüfer und Benutzer müssen über mehrere Verbindungen und Registerkarten gehen, um auf Ziele wie Webseiten und Ansichten zuzugreifen. Das Verständnis dieser Verbindungen und Registerkarten ist jedoch nicht entscheidend für die Bewertung.
- Terminplanungen werden mit begrenzter Automatisierung oder manuellen Planungen unterstützt, die regelmäßig überprüft werden müssen.

Auch wenn diese Schwächen aufgrund ihres Alters möglicherweise nicht mehr vollständig zutreffen, konnten keine neueren Studien gefunden werden, die einen aktuellen Vergleich der ALM-Tools und Anbieter bieten. Es ist daher weiterhin relevant, diese Schwächen bei der Betrachtung von ALM-Systemen zu berücksichtigen.

Zusammenfassend lässt sich ableiten, dass ALM-Tools verschiedener Anbieter das Lebenszyklusmanagement in der Entwicklungs- und Wartungsphase unterstützen. Ihre begrenzten Fähigkeiten sind an die unklare Definition von ALM gebunden und stark herstellerabhängig.

2.4 Natural Language Processing

NLP ist ein Teilgebiet der KI, das die Verarbeitung und Analyse menschlicher Sprache durch Computer ermöglicht und die Interaktion zwischen beiden erleichtert. Ziel von NLP ist es, Computern die Fähigkeit zu verleihen, gesprochene oder geschriebene menschliche Sprache zu verstehen, zu interpretieren und sinnvoll zu erzeugen. Dies umfasst eine Vielzahl von Aufgaben und Technologien, die in unterschiedlichen Anwendungen eingesetzt werden können. In diesem Kapitel wird der Begriff näher erläutert und in das übergeordnete Thema der KI eingeordnet. Zudem werden verschiedene allgemeine Anwendungsbereiche betrachtet und ein Überblick über die bestehenden Modelle gegeben.

2.4.1 Erläuterung und Einordnung des Begriffs Natural Language Processing

NLP ist ein Gebiet, das computergestützte Techniken zum Zweck des Lernens, Verstehens und Produzierens menschlicher Sprachinhalte einsetzt (Hirschberg und Manning 2015). Liddy (2001) lieferte diese Definition:

Die natürliche Sprachverarbeitung umfasst eine Reihe von Computertechniken, die darauf abzielen, natürlich vorkommende Texte auf verschiedenen linguistischen Ebenen zu analysieren und darzustellen, um menschenähnliche Sprachverarbeitung für verschiedene Aufgaben und Anwendungen zu ermöglichen (Liddy 2001, S.3).

In dieser Definition bezieht sich der Begriff der "Ebenen der linguistischen Analyse" auf die phonetische, morphologische, lexikalische, syntaktische, semantische, diskursive und pragmatische Analyse von Sprache (Liddy 2001), wobei davon ausgegangen wird, dass Menschen normalerweise alle diese Ebenen nutzen, um Sprache zu produzieren oder zu verstehen (Chowdhury 2003). NLP-Systeme können verschiedene Ebenen bzw. Kombinationen von Ebenen der Sprachanalyse unterstützen. Je mehr Analyseebenen NLP-Systeme unterstützen, desto stärker oder leistungsfähiger sollen diese Systeme sein (Cambria und White 2014).

Die Ansätze des NLP können gemäß Liddy (2001) in die Kategorie des symbolischen und des statistischen NLP unterteilt werden (Liddy 2001). Obwohl beide Arten von NLP seit den Anfängen des NLP-Feldes (etwa in den 1950er Jahren) untersucht wurden, dominierte bis in die 1980er Jahre das symbolische NLP das Feld.

Symbolisches NLP ist aus der KI hervorgegangen. Es basiert auf der expliziten Darstellung von Fakten über Sprache und zugehörigen Algorithmen und nutzt dieses Wissen zur tiefgreifenden Analyse linguistischer Phänomene . Zu den symbolischen NLP-Ansätzen gehören logische oder regelbasierte Systeme und semantische Netze. In regelbasierten Systemen wird linguistisches Wissen in Form von Fakten oder Produktionsregeln dargestellt, während in semantischen Netzen dieses Wissen als ein Netz miteinander verbundener Konzepte dargestellt wird.

Symbolischen NLP-Ansätzen mangelt es jedoch an der Flexibilität, sich dynamisch an neue Sprachphänomene anzupassen, da sie zur Analyse von Eingabetexten handschriftliche Regeln oder explizite Repräsentationen verwenden, die durch die menschliche Analyse von wohlformulierten Beispielen erstellt wurden (Liddy 2001). Solche Regeln können zu zahlreich werden, um sie zu verwalten (Nadkarni et al. 2011). Darüber hinaus können symbolische Ansätze bei unbekannten oder ungrammatischen Eingaben an ihre Grenzen stoßen (Bhatia et al. 2016). Seit den 1980er Jahren, vor allem aber in den 1990er Jahren, hat statistisches NLP wieder an Popularität gewonnen. Dieser Anstieg lässt sich durch die Verfügbarkeit kritischer Rechenressourcen und die Entwicklung von Methoden des maschinellen Lernens (ML) erklären (Hirschberg und Manning 2015). Seitdem ist statistisches NLP der Mainstream der NLP-Forschung und -Entwicklung (Cambria und White 2014). Zum Beispiel basieren viele der heutigen NLP-Tools wie Part-of-Speech-Tagger und syntaktische Parser auf statistischem NLP (Manning et al. 2014a).

Im Gegensatz zum symbolischen NLP werden beim statistischen NLP verschiedene ML-Methoden und große Mengen an linguistischen Daten (Textkorpora) verwendet, um approximative, probabilistische Modelle der Sprache zu entwickeln. Diese statistischen Modelle sind einfach und dennoch robust, da sie auf tatsächlichen Beispielen linguistischer Phänomene basieren, die von den Textkorpora geliefert werden, und nicht auf einer tiefgehenden Analyse linguistischer Phänomene wie im symbolischen NLP. Wenn statistisches NLP mit großen Mengen an annotierten Sprachdaten trainiert wird, kann es gute Ergebnisse erzielen, da es die häufigsten Fälle in den umfangreichen Daten lernen kann. Je umfangreicher und repräsentativer die Daten sind, desto besser wird statistisches NLP. Statistisches NLP kann jedoch auch bei unbekanntem oder fehlerhaftem Input nachlassen, ein ähnliches Problem wie beim symbolischen NLP. Der Hauptunterschied liegt jedoch darin, dass symbolisches NLP auf vorgegebenen Regeln und linguistischen Theorien basiert, während statistisches NLP auf Wahrscheinlichkeiten und Erfahrungen aus den Daten setzt. Dadurch ist statistisches NLP flexibler, aber auch anfälliger bei unzureichenden Daten (Nadkarni et al. 2011).

Darüber hinaus ist statistisches NLP hauptsächlich für Low-Level-NLP-Aufgaben wie lexikalische Erfassung, Parsing, Part-of-Speech-Tagging, Kollokationen und Grammatiklernen nützlich (Liddy 2001). Heute basieren viele Text- und Sentiment-Klassifikatoren des statistischen NLP immer noch ausschließlich auf der Verwendung der Wörter eines Textes, um die Bedeutung des Textes zu ermitteln, anstatt die Struktur und Semantik der Sätze oder Diskurse des Textes zu verwenden. Außerdem werden die meisten statistischen Modelle mit Textkorpora aus der Alltagssprache trainiert, wie Büchern oder Wikipedia Einträgen (Schramowski et al. 2021). Folglich kann statistisches NLP für domänenspezifische Texte wie Softwareanforderungen unzuverlässig sein. Seit etwa 2012 haben sich Methoden des Deep Learning (DL) in der NLP-Szene etabliert (Young et al. 2018). Die zentrale Idee von DL besteht darin, dass eine Maschine mit einer großen Menge an Rohdaten gefüttert werden kann und automatisch die für die Erkennung oder Klassifizierung benötigten Darstellungen oder Merkmale entdeckt (LeCun et al. 2015). Daher erfordert DL nur sehr wenig manuelles Feature-Engineering, bei dem Merkmale oder Eigenschaften der Daten, wie spezifische linguistische Muster, von Menschen händisch definiert und extrahiert werden müssen. Darüber hinaus sind die von DL-Modellen erlernten Merkmale hochrangig und ermöglichen eine bessere Generalisierung auch über neue, ungesehene Daten.

Im Gegensatz dazu sind herkömmliche ML-Techniken, die im NLP eingesetzt werden, nur begrenzt in der Lage, natürliche Daten in ihrer Rohform zu verarbeiten. Dies bedeutet, dass der Aufbau eines statistischen NLP-Systems eine sorgfältige technische Planung und beträchtliche Fachkenntnisse erfordert, um einen Merkmalsextraktor zu entwickeln, der den Rohtext in eine geeignete interne Darstellung (d. h. einen Merkmalsvektor) umwandelt, anhand derer das ML-Teilsystem, häufig ein Textklassifikator, Muster in der Eingabe erkennen oder klassifizieren kann. Sowohl NLP- als auch Deep-Learning-Experten prognostizierten, dass NLP ein Bereich ist, in dem Deep Learning in den nächsten Jahren einen großen Einfluss haben würde (Hirschberg und Manning 2015; LeCun et al. 2015). Nichtsdestotrotz zeigen Trends im auf Deep Learning basierenden NLP, dass die Kopplung von symbolischer KI der Schlüssel zum Fortschritt auf dem Weg vom NLP zum Verständnis natürlicher Sprache ist (Young et al. 2018). Dies bestätigt die These, dass symbolische Ansätze und statistische Ansätze komplementär sind.

Aufbauend auf den ermittelten Grundlagen zu ALM und NLP werden die Anwendungsmöglichkeiten von NLP im ALM untersucht. Um diese fundiert zu erforschen, wird im nächsten Schritt eine systematische Literaturrecherche durchgeführt, die bestehende Ansätze und Technologien identifizieren soll, die zur Optimierung von ALM-Prozessen beitragen können.

2.4.2 Anwendungen von Natural Language Processing

Im Allgemeinen können die Anwendungsbereiche des NLP in acht Kategorien aufgeteilt werden, wie in Abbildung 2-6 dargestellt ist. Diese werden in diesem Kapitel jeweils kurz erklärt.

Spracherkennung (Speech Recognition) ist ein grundlegender Anwendungsbereich von NLP und befasst sich mit der Umwandlung gesprochener Sprache in Text. Diese Technologie findet Anwendung in virtuellen Assistenten wie Siri oder Alexa, die gesprochene Befehle verstehen und ausführen können. Sie ermöglicht auch Spracherkennungssysteme in Smartphones und Autos, die es uns erlauben, freihändig zu kommunizieren oder Navigationseingaben zu machen (Hirschberg und Manning 2015; Manning et al. 2014a).

NLP Anwendungsbereiche
Spracherkennung
Sprachgenerierung
Textklassifikation
Sentimentanalyse
Informationsextraktion
Frage-Antwort-Systeme
Textzusammenfassung
Maschinelle Übersetzung

Abbildung 2-6 Anwendungsbereiche von Natural Language Processing

Sprachgenerierung (Natural Language Generation) hingegen ist das Gegenstück zur Spracherkennung. Hierbei geht es darum, basierend auf Daten oder vordefinierten Regeln menschlich klingende Texte zu erzeugen. Diese Technologie wird beispielsweise in der automatischen Erstellung von Berichten, der Generierung von personalisierten Nachrichten oder in Chatbots verwendet, die auf natürliche Weise mit Nutzern interagieren können. Die Qualität der generierten Texte hängt stark von der zugrundeliegenden Datenbasis und den verwendeten Algorithmen ab (Young et al. 2018; Schramowski et al. 2021).

Ein weiterer Anwendungsbereich ist die Textklassifikation. Diese Methode sortiert Texte automatisch in vordefinierte Kategorien ein. Ein klassisches Beispiel hierfür ist die Filterung von E-Mails in „Spam“ und „Nicht-Spam“ oder die thematische Sortierung von Dokumenten und Nachrichtenartikeln. Die Textklassifikation wird auch verwendet, um in sozialen Medien Beiträge zu bestimmte Themen oder Meinungen zu identifizieren und zu gruppieren (Zhao et al. 2022). Die Sentimentanalyse baut auf der Textklassifikation auf und geht darüber hinaus, indem Emotionen und Einstellungen in einem Text identifiziert und bewertet werden. Dies wird insbesondere im Bereich des Marketings und des Kundenfeedbacks genutzt, um zu erfassen, wie Produkte oder Dienstleistungen von der Öffentlichkeit wahrgenommen werden. Durch die Analyse von Rezensionen, Tweets oder anderen Online-Äußerungen kann auf negative Stimmungen zeitnah reagiert und entsprechende Anpassungen der Strategien vorgenommen werden (Zhao et al. 2022). Informationsextraktion zielt darauf ab, relevante und strukturierte Informationen aus unstrukturierten Texten zu extrahieren. Dabei kann es sich um das Erkennen von Namen, Orten, Daten oder spezifischen Ereignissen in einem Text handeln. Diese Technologie wird in der juristischen Recherche, in Nachrichtenaggregatoren oder in der

medizinischen Dokumentation eingesetzt, um relevante Informationen schnell und effizient zu extrahieren und weiterzuverarbeiten (Zhao et al. 2022).

Ein weiterer Anwendungsbereich von NLP sind Frage-Antwort-Systeme (Question Answering). Diese Systeme sind darauf ausgelegt, auf natürliche Sprachfragen präzise und relevante Antworten zu geben. Sie finden Anwendung in Suchmaschinen, wo sie den Nutzern gezielte Antworten auf ihre Anfragen liefern, oder in Chatbots, die Kundenanfragen beantworten. Diese Systeme nutzen oft große Datenbanken und fortschrittliche Algorithmen, um die richtige Antwort aus einer Vielzahl von Möglichkeiten auszuwählen (Khurana et al. 2023). Die Textzusammenfassung (Summarization) ist eine weitere Anwendung, bei der lange Texte automatisch auf ihre wesentlichen Inhalte reduziert werden. Dies ist besonders nützlich in Bereichen, in denen große Mengen an Texten verarbeitet werden müssen, wie in Nachrichtenredaktionen, wissenschaftlichen Publikationen oder im juristischen Bereich. Durch die Textzusammenfassung können Benutzer schnell die wichtigsten Punkte eines Dokuments erfassen, ohne den gesamten Text lesen zu müssen (Hirschberg und Manning 2015; Young et al. 2018).

Schließlich spielt die maschinelle Übersetzung im globalen Kontext eine Rolle. Sie ermöglicht die automatische Übersetzung von Texten von einer Sprache in eine andere und trägt damit wesentlich zur Überwindung von Sprachbarrieren bei. Maschinelle Übersetzungssysteme wie Google Translate nutzen fortschrittliche NLP-Techniken, um nicht nur Wort-für-Wort-Übersetzungen zu liefern, sondern auch den Kontext und die Feinheiten der Ausgangssprache zu berücksichtigen, um eine möglichst präzise und natürliche Übersetzung zu erreichen (Hirschberg und Manning 2015). Dieser Anwendungsbereich ist für ALM allerdings nicht relevant, weshalb er hier nicht weiter betrachtet wird.

Insgesamt kann zusammengefasst werden, dass NLP eine Vielzahl von Technologien umfasst, die es Maschinen ermöglicht, menschliche Sprache auf vielfältige Weise zu verarbeiten und zu nutzen. Diese Technologien haben das Potenzial, zahlreiche Bereiche unseres täglichen Lebens zu verbessern und zu transformieren, von der Art und Weise, wie wir mit Technologie interagieren, bis hin zur Verbesserung der Effizienz in verschiedenen Branchen.

2.4.3 Vorstellung einer Auswahl von Modellen

Es gibt verschiedene Arten von NLP-Modellen, die jeweils für unterschiedliche Aufgaben und Anwendungsbereiche entwickelt wurden. Diese Modelle lassen sich grob in mehrere Kategorien einteilen, basierend auf ihrer Architektur und ihrem Anwendungszweck.

Sprachmodelle

Diese Modelle sind darauf ausgelegt, natürlichen Text zu generieren und zu verstehen. Sie basieren in der Regel auf Transformer-Architekturen und können menschenähnliche Texte erstellen, Konversationen führen, Texte vervollständigen und in manchen Fällen sogar spezifische Aufgaben wie die Beantwortung von Fragen lösen. Da viele bekannte Modelle als Open-Source-Versionen verfügbar sind, können sie weiter verbessert oder für spezialisierte Anwendungen angepasst werden, wodurch eine Vielzahl von Sprachmodellen auf dem Markt vorhanden ist. Eine zentrale Plattform in diesem Bereich ist Hugging Face, die als Bibliothek und Open-Source-Plattform für Maschinelles Lernen und NLP bekannt ist. Sie stellt Entwicklern vortrainierte Modelle, Datensätze und Tools zur Verfügung, um den Zugang zu modernster KI-Technologie zu erleichtern. Auf Hugging Face sind beispielsweise 183 Open-Source-Sprachmodelle verzeichnet. Hier wird nur ein kleiner Ausschnitt von bekannten Modellen aufgeführt:

Tabelle 2-2 Übersicht aktueller Sprachmodelle

Modell	Herausgeber	Parameter	Anwendungszweck
GPT-4o	OpenAI (OpenAI 2024b)	> 175 Mrd.	Natürliche Sprache verstehen und generieren, Textvervollständigung, Konversationssysteme (z.B. ChatGPT) und komplexe Textverarbeitung für Fragebeantwortung, Übersetzung, und Textzusammenfassung.
GPT-4o mini	OpenAI (OpenAI 2024a)	> 175 Mrd.	Leichtere, effizientere Version von GPT-4o für ähnliche Aufgaben wie Textverarbeitung, Übersetzung und Textgenerierung, jedoch mit reduzierten Rechenanforderungen.
Gemini 1.5	Google (Pichai und Hassabis 2024)	1.8 / 3.25 Mrd.	Multimodales KI-Modell, das Text und visuelle Daten kombiniert, um Aufgaben wie Bildbeschreibung, Bild-Text-Generierung und Textanalyse durchzuführen.
Claude	Anthropic (Anthropic 2024)	~ 150 Mrd.	Sprachgenerierung mit Fokus auf ethische und kontrollierte KI-Anwendungen, optimiert für Dialogsysteme, Textgenerierung und Assistenzsysteme.
Command	Cohere (Cohere 2024)	35 Mrd.	NLP-Modell spezialisiert auf Befehlsinterpretation und -ausführung in konversationellen Systemen, z.B. für Chatbots und interaktive Sprachassistenten.
GPT-Neo	EleutherAI (EleutherAI 2023)	1.37 Mrd.	Open-Source-Alternative zu GPT-3, verwendet für Textgenerierung, Dialogsysteme, maschinelle Übersetzung und verschiedene NLP-Aufgaben wie Textklassifikation.
BERT	Google (Devlin et al. 2018)	110 Mio.	Sprachverständnis, besonders bei Aufgaben wie Fragebeantwortung, Textklassifikation, Named Entity Recognition (NER), Textzusammenfassung.
RoBERTa	FacebookAI, jetzt Meta (Liu et al. 2019)	125 Mio.	Verbesserte Version von BERT, optimiert für allgemeine NLP-Aufgaben wie Textklassifikation, Fragebeantwortung und Sentimentanalyse, mit einer stärkeren Performance.
DistilBERT	Hugging Face (Sanh et al. 2019)	69 Mio.	Leichtere, effizientere Version von BERT, optimiert für NLP-Aufgaben, die geringere Ressourcen erfordern, aber immer noch eine hohe Genauigkeit

			benötigen (z.B. Textklassifikation, Fragebeantwortung).
Sentence-BERT	Hugging Face (Reimers und Gurevych 2019)	4.39 Mio.	Speziell für die semantische Ähnlichkeitsbewertung von Sätzen, Textpaaren oder Satzvektoren optimiert. Wird für Aufgaben wie Textsuche, Clustering oder Textmatching verwendet.
T5	Google (Google 2022)	223 / 738 Mio.	Einheitliche Text-zu-Text-Darstellung von NLP-Aufgaben wie Textklassifikation, Textzusammenfassung, Fragebeantwortung, maschinelle Übersetzung.
Phi-3	Microsoft (Bilenko 2024)	3.8 / 7 / 14 Mrd.	KI-Modelle für allgemeine Sprachverständnis- und Sprachgenerierungsaufgaben, oft im Bereich der Forschung verwendet, ähnlich GPT-ähnlichen Modellen.
LLaMA 3.1	Meta (Meta 2024)	8 / 70 / 405 Mrd.	Open-Source-Sprachmodell von Meta, für natürliche Sprachgenerierung, Sprachverständnis, Dialogsysteme und ähnliche Nicht für kommerzielle Nutzung als Open Source freigegeben.
BART	Facebook, jetzt Meta (Lewis et al. 2019)	139 / 400 Mio.	Generative NLP-Aufgaben wie Textzusammenfassung, maschinelle Übersetzung und Textwiederherstellung (z.B. Fehlerkorrektur).
Mistral 7B	MistralAI (Mistral AI 2024)	7.24 Mrd.	Transformer-Modell für Aufgaben der Sprachgenerierung und -verarbeitung, entwickelt mit einem Fokus auf optimierte Rechenleistung und Effektivität für NLP-Aufgaben wie Textgenerierung und maschinelle Übersetzung.

Tabelle 2-2 bietet eine Übersicht über eine Auswahl an Sprachmodellen, die von verschiedenen Organisationen entwickelt wurden. Die Auswahl umfasst sowohl kommerziell vertriebene als auch Open-Source-Modelle. Die Tabelle enthält Angaben zu den Herausgebern, der Anzahl der Parameter sowie dem jeweiligen Anwendungszweck. Modelle wie GPT-4o (OpenAI 2024b) und GPT-4o mini (OpenAI 2024a) von OpenAI basieren auf der Transformer-Architektur und verfügen über eine hohe Anzahl an Parametern, die für Aufgaben der Sprachgenerierung und des Sprachverständnisses genutzt werden. Kleinere Modelle wie DistilBERT (Sanh et al. 2019) und Sentence-BERT (Reimers und Gurevych 2019) wurden entwickelt, um effizientere Lösungen mit geringeren Rechenanforderungen zu bieten. Ebenfalls enthalten sind multimodale Modelle wie Gemini 1.5 von Google (2022), die Text- und Bilddaten kombinieren, sowie spezialisierte Modelle wie Command (Cohere 2024), das auf die Befehlsinterpretation in konversationellen Systemen ausgerichtet ist. Weitere Modelle wie BERT (Devlin et al. 2018), RoBERTa (Liu et al. 2019) und T5 (Google 2022) decken ein breites Spektrum an NLP-Aufgaben ab, einschließlich Textklassifikation, Fragebeantwortung und Textzusammenfassung. Modelle wie LLaMA 3.1 von Meta (2024) und Mistral 7B (Mistral AI 2024) zeichnen sich durch ihren Open-Source-Charakter und eine fokussierte Entwicklung auf effiziente Rechenleistung aus.

Bibliotheken und Frameworks

Weiterhin gibt es NLP Bibliotheken und Frameworks, die als umfassende Sammlungen von Werkzeugen, Funktionen und Algorithmen, die für die Verarbeitung natürlicher Sprache entwickelt wurden. Sie bieten grundlegende Funktionen wie Tokenisierung, Stemming, Lemmatisierung, Named Entity Recognition (NER), Textklassifikation und vieles mehr. Diese Ansätze decken eine Vielzahl von NLP-Aufgaben ab und umfassen häufig klassische Algorithmen und Methoden, die nicht zwingend auf DL basieren. Häufig in der Wissenschaft verwendete Beispiele sind:

Tabelle 2-3 Übersicht von Natural Language Processing Bibliotheken und Frameworks

Nr	Bibliothek /Framekwork	Herausgeber	Programmiersprache	Anwendungszweck
1	SpaCy	Explosion AI (Explosion 2024)	Python	Vortrainierte Modelle für Sprachverarbeitung, Aufgaben wie Named Entity Recognition und Textklassifikation.
2	Scitkit-learn	Open Source Community (Pedregosa et al. 2011)	Python	Werkzeug für maschinelles Lernen, inklusive Klassifikation, Regression und Clustering.
3	TensorFlow	Google (TensorFlow 2024)	Python, C++	Bibliothek für maschinelles Lernen, mit Schwerpunkt auf Deep Learning, skalierbare Modelle und neuronale Netze.
4	PyTorch	Meta (PyTorch 2024)	Python	Bibliothek für maschinelles Lernen, spezialisiert auf Deep Learning und neuronale Netze.
5	Stanford CoreNLP	Stanford University (Manning et al. 2014b)	Java	Breites Spektrum von NLP-Tools, einschließlich Parsing und Named Entity Recognition.
6	Natural Language Toolkit (NLTK)	Open Source Community (NLTK Project 2024)	Python	Bildungszwecke und Prototypisierung von NLP-Anwendungen.

Tabelle 2-3 bietet eine Übersicht über zentrale Bibliotheken und Frameworks, die für NLP und ML verwendet werden. Die Auswahl der Bibliotheken wurde auf Basis ihrer weit verbreiteten Nutzung in der Forschung und Entwicklung sowie ihrer Vielseitigkeit getroffen. Die aufgenommenen Bibliotheken decken eine Vielzahl von Anwendungsbereichen ab, von vortrainierten Modellen für spezifische NLP-Aufgaben bis hin zu umfassenden Frameworks für DL. Diese Bibliotheken und Frameworks werden häufig verwendet, um Daten für Sprachmodelle vorzubereiten und vorzuverarbeiten. Bibliotheken wie SpaCy, Stanford CoreNLP und NLTK sind speziell darauf ausgelegt, Vorverarbeitungsaufgaben in NLP zu erleichtern. TensorFlow und PyTorch hingegen werden häufig für das Training und die Optimierung der eigentlichen Sprachmodelle verwendet, nachdem die Daten mit den NLP-

Bibliotheken vorverarbeitet wurden. Scikit-learn kann auch zur Datenvorbereitung und -analyse verwendet werden, insbesondere bei klassischen maschinellen Lernansätzen.

3 Systematische Literaturrecherche zur Ermittlung von Anwendungsmöglichkeiten von Natural Language Processing im Application Lifecycle Management

Dieses Kapitel gibt einen umfassenden Überblick über die aktuellen Technologien, identifiziert relevante Anwendungsbereiche und -möglichkeiten und zeigt potenzielle Forschungslücken sowie zukünftige Forschungsfelder auf. Dabei wird darauf geachtet, dass die erhobenen Daten valide, relevant und umfassend sind. In den folgenden Abschnitten wird zunächst die gewählte Methodik aufgeschlüsselt und erläutert, anschließend die Ergebnisse der Literaturrecherche aufgezeigt und diskutiert.

3.1 Methodik der systematischen Literaturrecherche

Es gibt verschiedene Ansätze für die Durchführung einer systematischen Literaturrecherche (SLR). vom Brocke et al. (2009) erarbeiteten einen Ansatz, um die wissenschaftliche Rigorosität sicherzustellen. Sie stellen heraus, dass eine SLR eine Zirkularität, wie in Abbildung 3-1 dargestellt ist, besitzt. Es ist folglich zu beachten, dass die Ergebnisse einer SLR nur eine Momentaufnahme der aktuellen Forschungs- bzw. Entwicklungslage darstellt, aber das Wissen ständig vergrößert wird, wodurch Literaturübersichten nach einer gewissen Zeit oft veraltet sind und Anlass zu einer Erweiterung und Aktualisierung der Sichtweise geben (Pervan 1998, S. 158).

Abbildung 3-1 Rahmen für die Literaturrecherche (vom Brocke et al. (2009, S.2261))

Eine große Herausforderung bei der Überprüfung der Literatur besteht darin, einen angemessenen Umfang der Überprüfung zu definieren. Rezensionen können kritisch, interpretierend, spekulativ, auf dem neuesten Stand der Forschung und historisch sein und können in Bezug auf den Gegenstand, den behandelten Zeitraum und den Grad der Abdeckung der Quellen variieren (Garfield 1987, S. 114). Darüber hinaus können Literaturübersichten einem breiten Spektrum von zum Teil sehr unterschiedlichen Zwecken dienen, die von der Gewinnung neuer und der Synthese bestehender Forschungsergebnisse bis hin zur Identifizierung von Forschungsmethoden oder -techniken reichen, die in einem Bereich häufig verwendet werden (Hart 1998).

Die Transparenz und Reproduzierbarkeit der Ergebnisse ist ein wichtiges Merkmal für eine SLR, weshalb die rigorose Umsetzung eines Ansatzes unerlässlich ist. Dafür wird im Rahmen dieser Arbeit die Vorgehensweise von Sinzig (2017, S.69) verwendet.

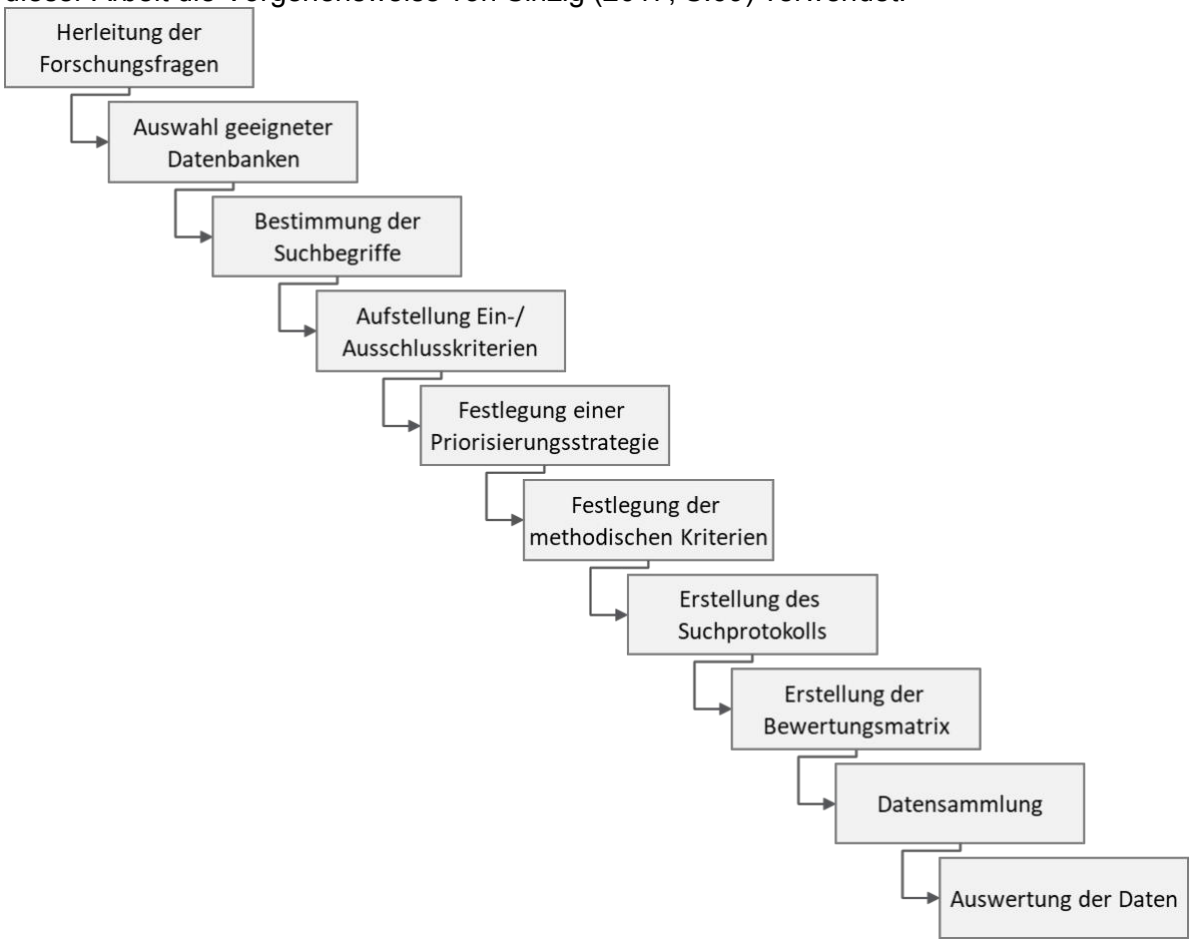


Abbildung 3-2 Vorgehensweise einer systematischen Literaturanalyse [Gerdt (2023, S.19) und Sinzig (2017, S.69)]

Dafür sollte im ersten Schritt die Formulierung der Forschungsfrage oder -fragen stattfinden. Eine klare Definition der Forschungsfrage dient dazu, die Relevanz und den Umfang der Recherche zu bestimmen. Als nächstes müssen die für das Thema geeigneten Datenbanken ermittelt und die passenden Suchbegriffe bestimmt werden. Zur Definition des Rahmens der Analyse ist es notwendig, Ein- und Ausschlusskriterien festzulegen. Darüber hinaus sollten methodische Kriterien entwickelt werden, nach denen die Quellen systematisch geordnet und bewertet werden. Im Anschluss daran erfolgt die Datensammlung und Auswertung der relevanten Datenbanken.

Gerdt (2023) hat Sinzigs Ansatz durch vier Schritte erweitert. Zusätzlich zu den Kriterien wird eine Priorisierungsstrategie festgelegt, um eine systematische und effiziente Auswahl relevanter Ergebnisse sicherzustellen. Selbst bei klar definierten Ein- und Ausschlusskriterien kann die Menge der gefundenen Literatur so umfangreich sein, dass eine manuelle Sichtung den vorgegebenen Zeitrahmen eines Projektes überschreiten würde. Daher wird eine Priorisierungsstrategie angewendet, um die relevantesten Ergebnisse zu identifizieren und zu analysieren. Zur weiteren Strukturierung dienen die Erstellung eines Suchprotokolls, einer Bewertungsmatrix, sowie der Erstellung eines Anforderungsprofils zur Bewertung der Quellen. Da die Erstellung des Anforderungsprofils themenspezifisch hinzugefügt wurde und für das ermittelten Forschungsfragen nicht notwendig ist, wird dieser Schritt weggelassen. Diese Vorgehensweise ist in Abbildung 3-2 abgebildet. Die Forschungsfragen wurden in Kapitel 1 beschrieben, die Analyse gilt zur Beantwortung der Forschungsfrage 1. Die weiteren Schritte der SLR werden in den folgenden Abschnitten erarbeitet.

3.2 Festlegung der Parameter

Bestandteile der Konzeptionalisierung des Themas sind die Auswahl der geeigneten Parameter, die für die systematische Durchführung der Literaturrecherche benötigt werden. Dafür werden im Folgenden die im Kapitel 3.1 aufgestellten Schritte zwei bis sieben durchgeführt und erläutert.

3.2.1 Festlegung der Datenbanken

Die Nutzung mehrerer Datenbanken und umfassender Suchstrategien, einschließlich Boolescher Operatoren und Trunkierungen werden genutzt, um relevante Literatur zu identifizieren. Laut Wyrwich et al. (2024, S.2768) sind die wichtigsten Datenbanken für Literatur zum Systems Engineering die folgenden:

- Scopus
- Web of Science
- IEEE Xplore

ALM und Systems Engineering (SE) sind zwei Disziplinen, die eng miteinander verbunden sind und sich gegenseitig ergänzen, insbesondere in der Entwicklung komplexer Systeme und Softwarelösungen. ALM fokussiert sich auf den gesamten Lebenszyklus von Softwareanwendungen, während SE systematische Ansätze zur Entwicklung technischer Systeme bietet. Da beide Ansätze in der Praxis oft miteinander kombiniert werden, um komplexe Software- und Hardwaresysteme effizient zu entwickeln, sollen diese Datenbanken für die SLR verwendet werden.

Das Feld des NLP verändert sich sehr schnell aufgrund einer Kombination aus technologischen Fortschritten, wachsender Datenverfügbarkeit und interdisziplinären Entwicklungen (Khurana et al. 2023). Deher ist zu beachten, dass diese Analyse nur einen Einblick in das aktuelle Forschungsfeld gibt.

3.2.2 Festlegung der Suchbegriffe und der Suchstrings

Ein Weg zur Identifizierung von Schlüsselbegriffen stellt das Concept Mapping dar, das auch die Möglichkeit bietet, relevante Suchbegriffe (insbesondere verwandte Begriffe oder Synonyme und Homonyme) aufzudecken, die bei der anschließenden Literaturrecherche verwendet werden können (Rowley und Slack 2004). Dies wird in den Kapiteln 2.3 und 2.4 aufgezeigt, wodurch mögliche Search Strings für die Datenbanken ermittelt werden können. Vor der Festlegung auf einen Search String werden verschiedene Optionen erarbeitet und getestet, und der folgende als der Search String mit der höchsten Eignung identifiziert:

Tabelle 3-1 Festlegung der Schlüsselbegriffe (Eigene Darstellung in Anlehnung an Gerdt (2023, S.20))

F1: „*Wie kann Natural Language Processing wertschöpfend für Application Lifecycle Management Anwendungen genutzt werden?*“		
Aspekt und Einordnung	**Aspekt 1** (Natural Language Processing)	**Aspekt 2** (Application Lifecycle Management)
Synonyme und artverwandte Begriffe in Englisch	Natural Language Processing	Application Lifecycle Management Requirements Engineering Document Management Test Management Bug and Issue Management

Die in Kapitel 1 aufgestellte Forschungsfrage 1 beinhaltet die Schlüsselbegriffe „Natural Language Processing“, sowie „Application Lifecycle Management“. Wie in Kapitel 2 behandelt, kann dieses in Unteraufgaben, die in Abbildung 2-4 zu finden sind, aufgeteilt werden. Die Gebiete, in denen die Anwendungsbereiche von NLP angewandt werden können, da natürliche Sprache vorkommt sind das Anforderungsmanagement, das Dokumentenmanagement, das Testmanagement, sowie die Fehler- und Problemverfolgung. In einer vorab Überprüfung der Datenbanken mit Stichwörtern konnten nicht viele Treffer mit „Natural Language Processing“ zusammen mit „Application Lifecycle Management“ gefunden werden, weshalb die Unterkategorien in den Suchstring mit einbezogen werden.

Die Begriffe aus Tabelle 3-1 werden zusammen mit booleschen Operatoren und Wildcards zusammen zu einem Suchstring zusammengehangen, um die Datenbanken und Blogs zu durchsuchen. Zu den häufigsten booleschen Operatoren gehören AND, OR und NOT. Der AND-Operator wird verwendet, um Suchergebnisse zu finden, die alle angegebenen Begriffe enthalten, wodurch die Ergebnismenge eingeschränkt wird. Der OR-Operator hingegen erweitert die Suche, indem er Ergebnisse zurückgibt, die mindestens einen der angegebenen Begriffe enthalten. Der NOT-Operator wird verwendet, um bestimmte Begriffe aus den Suchergebnissen auszuschließen, wodurch die Ergebnismenge weiter eingeschränkt wird (Scopus 2024).

Neben den booleschen Operatoren spielen auch Wildcards eine wichtige Rolle bei der Verfeinerung von Suchanfragen. Das Sternchen (*) ist eine der am häufigsten verwendeten Wildcards und ersetzt null oder mehr Zeichen in einem Wort. Es ist nützlich, um verschiedene Formen oder Varianten eines Begriffs zu finden. Ein Fragezeichen (?) ersetzt genau ein Zeichen und wird verwendet, um nach Wörtern zu suchen, die sich nur durch ein Zeichen unterscheiden. Anführungszeichen (" ") können verwendet werden, um eine exakte Phrase zu suchen, wobei die Wörter in genau der angegebenen Reihenfolge erscheinen müssen. Die Verwendung dieser booleschen Operatoren und Wildcards ermöglicht es, Suchanfragen präziser zu gestalten und relevantere Ergebnisse zu erzielen (Scopus 2024).

Nach Prüfung der Eignung durch die oberflächliche Sichtung der Ergebnisse in den Datenbanken resultiert der folgende Suchstring zur Beantwortung von F1:

Suchstring: “Natural language processing” AND (“Application Lifecycle Management” OR “Requirements Engineering” OR “Document Management” OR “Test Management” OR “Bug and Issue Management”)*

Nachdem der Suchstring festgelegt ist, müssen die Ein- und Ausschlusskriterien erfasst werden, um den Suchprozess gezielt einzugrenzen und die Effizienz der Suche zu optimieren.

3.2.3 Festlegung der Ein- und Ausschlusskriterien

Nach einer ersten Sichtung der Titel und Abstracts der anhand des oben erarbeiteten Suchstrings gefundenen Literatur zeigt sich, dass deutschsprachige Artikel zu dem Thema vernachlässigt werden können. Dies liegt zum einen daran, dass innovative Forschung zu dem Thema in der Regel auf Englisch veröffentlich wird, zum anderen auch, dass die relevanten Technologie-Blogs englischsprachig sind. Durch die schnellen technologischen Fortschritte wird der Erfassungszeitraum für die Quellen zudem eng bemessen, da das Feld durch die Veröffentlichung des Sprachmodells GPT im Jahr 2022 stark verändert wurde. Daher wird der Erfassungszeitraum auf die Jahre 2023 und 2024 begrenzt.

3.2.4 Festlegung der Priorisierungsstrategie

Um die Literaturrecherche weiter zu strukturieren, wird eine Priorisierungsstrategie nach Gioia et al. (2013) und vom Brocke et al. (2009) angewandt. Dabei folgt der Schlagwörtersuche in den Datenbanken und Blogs eine erste manuelle Prüfung, bei der die Abstracts der gefilterten Studien gesichtet und eindeutig irrelevante Studien ausgeschlossen werden. Gleichzeitig werden doppelte Studien identifiziert und entfernt, um Redundanzen zu vermeiden. In der

anschließenden Relevanzbewertung wird jede Studie anhand eines Relevanz-Scores beurteilt, der auf mehreren Faktoren wie thematischer Nähe, Zielbranche und Art der NLP Anwendung basiert. Dieser wird in Kapitel 3.3 mit der Bewertungsmatrix weiter erläutert. Zusätzlich werden die Studien thematisch gruppiert, wobei nur die relevantesten Gruppen für die weitere Analyse berücksichtigt werden. In der endgültigen Auswahl werden Studien, die hohe Relevanzbewertungen aufweisen, in die finale Analyse einbezogen. Dabei wird darauf geachtet, dass die ausgewählten Studien unterschiedliche methodische Ansätze, Perspektiven und Forschungsergebnisse umfassen, um eine fundierte Darstellung des Forschungsthemas zu gewährleisten.

3.2.5 Festlegung der methodischen Kriterien

Methoden können in qualitative, quantitative oder kombinierte Mixed-Method-Ansätze unterteilt werden (Sinzig 2017). Die Wahl der Methode wird dabei maßgeblich durch die Forschungsfrage und den Untersuchungsbereich bestimmt. Die qualitative Methodik zielt darauf ab, textbasierte Inhalte zu extrahieren und diese systematisch darzustellen. Im Gegensatz dazu zielen quantitative Methoden darauf ab, numerische Daten zu erheben und zu analysieren, um statistische Muster und Zusammenhänge zu identifizieren. Diese Art der Untersuchung, als Meta-Analyse bekannt, stellt keine eigenständige Methode dar, sondern ergänzt die systematische Literaturanalyse im Kontext quantitativer Literaturauswertung (Döring 2023). Der Mixed-Method-Ansatz wird verwendet, wenn eine strikte Trennung zwischen quantitativer und qualitativer Literatur nicht möglich ist oder wenn beide Ansätze für das Forschungsfeld relevant sind. Das Ziel dieser Arbeit besteht darin, einen Überblick über die aktuelle Forschungslage und Anwendungsmöglichkeiten von NLP im ALM zu erlangen, wobei der Mixed-Method Ansatz verfolgt wird. Der quantitative Teil nimmt nicht das Ausmaß einer Meta-Analyse an, jedoch werden die Ergebnisse nicht nur qualitativ, sondern auch quantitativ ausgewertet werden.

Nach der Festlegung der Suchparameter erfolgt die systematische Literaturanalyse. Im Vorfeld werden die Grundstruktur der erforderlichen Dokumentation sowie die Bewertungskriterien entwickelt.

3.3 Suchprotokoll und Bewertungsmatrix

Zu Beginn der SLR werden ein Suchprotokoll und eine Bewertungsmatrix erstellt. Im Suchprotokoll werden die relevanten Informationen aus den identifizierten Quellen systematisch erfasst, während die Bewertungsmatrix diese Quellen nach ihrer Relevanz für die Forschungsfrage strukturiert.

3.3.1 Erstellung des Suchprotokolls

Das Führen eines Suchprotokolls während der Literaturrecherche ist empfehlenswert, um auch bei zeitlicher Distanz einen umfassenden Überblick zu gewährleisten. Zudem ermöglicht es im nachfolgenden Prozess eine transparente Darstellung sowie Begründung der erzielten Resultate. Ferner können im Verlauf der Literaturrecherche Anpassungen an den vorab festgelegten Parametern vorgenommen werden, welche durch das Protokoll schlüssig begründet werden können. Die Konzeption des Protokolls ist dabei nicht vorab festgelegt, sondern kann individuell an den jeweiligen Anwendungsfall angepasst werden (Gerdt 2023).

Für den vorliegenden Anwendungsfall wird ein Protokoll erstellt, welches die verwendete Datenbank, das Themengebiet, den Dokumenttypen, den Erfassungsbereich, die Anzahl der gesamten Suchergebnisse, sowie die Anzahl der einbezogenen Ergebnisse beinhaltet. Die Übersicht ist in Tabelle 3-2 dargestellt.

Tabelle 3-2 Vorlage Suchprotokoll (Eigene Darstellung in Anlehnung an vom Brocke et al. (2009, S.7) und Gerdt (2023, S.25))

Datenbank	Erfassungs-bereich	Anzahl der Suchergebnisse	Anzahl der einbezogenen Ergebnisse nach Sichtung des		
			Titels	Abstract	Inhalt
Datenbank 1	2022-2024	100	70	40	20

3.3.2 Erstellung der Bewertungsmatrix

Als letzten Schritt vor der Synthese wird die Bewertungsmatrix ermittelt, anhand dessen die Relevanz der einbezogenen Quellen bewertet wird. Dafür werden weiterhin die Vorgehensweisen von vom Brocke et al. (2009) und Gerdt (2023) kombiniert und angewandt. vom Brocke et al. (2009) betonen in ihrer Arbeit die Bedeutung methodischer Strenge bei der Dokumentation des Literaturrechercheprozesses. Dies umfasst klare Kriterien für die Bewertung und Auswahl von Literaturquellen, um die Nachvollziehbarkeit und Replizierbarkeit des Suchprozesses sicherzustellen. In der Bewertungsmatrix wird der methodische Ansatz widergespiegelt, indem spezifische Kategorien und Symbole verwendet werden, um Konzepte systematisch zu bewerten und zu dokumentieren. Es wird eine klare Struktur angewendet, in der zwischen vollständig vorhandenen, teilweise behandelten und kaum behandelten Konzepten unterschieden wird.

Die Struktur der Bewertungsmatrix von Gerdt (2023, S.26) wird verwendet, wie zu sehen in Tabelle 3-3 und die beschriebenen Bewertungskriterien von vom Brocke et al. (2009, S.12) übernommen. Dafür müssen zunächst die gesuchten Konzepte identifiziert werden. Dies wirft die Frage auf, welche Informationen extrahiert werden sollen. Ein wichtiges Thema dabei sind die Tools, Bibliotheken und Methoden mit denen NLP angewandt wird. Zusätzlich ist zu bestimmen, in welchem Feld des Application Lifecycle Managements wird NLP angewandt, und welcher genaue Use Case, mit welcher Art von NLP umgesetzt wird. Noch einmal zur Übersicht werden Informationen in den folgenden Konzepten gesammelt:

- NLP Tool/Bibliothek/Methode
- NLP Anwendungsart
- ALM Lösung
- Use Case

Das Vorkommen aller dieser Konzepten sind die Kriterien, nach denen die Eignung der Quellen bewertet werden soll.

El-Hajjami et al. (2023) stellte fest, dass Machine Learning Methoden im Gegensatz zu Sprachmodellen speziell auf die Daten trainiert werden müssen, um eine gute Performance abzuliefern, wogegen Sprachmodelle dies nicht benötigen. Aus diesem Grund sollen in diese Recherche in erster Linie Sprachmodelle analysiert werden. Etwaige Kombinationen von Methoden sollen zusätzlich berücksichtigt werden. Des Weiteren ist die Zielgruppe dieser Auswertung die Industrie, weshalb medizinische und juristische Quellen ausgeschlossen werden.

Tabelle 3-3 Beschreibung der Bewertungsmatrix (Eigene Darstellung in Anlehnung an Gerdt (2023, S.26) und vom Brocke et al. (2009, S.12))

Kategorie	Symbol	Beschreibung
1	++	Alle Konzepte sind vorhanden
2	+	Einige der Konzepte sind vorhanden und werden behandelt
3	-	Wenige der Konzepte sind vorhanden und werden behandelt

Die in Tabelle 3-3 angeführte Analogie dient dazu, das Erfüllungsmaß der einzelnen Konzepte des Modells zu bewerten. Eine vollumfängliche Erfüllung der Anforderung wird durch ein doppeltes Pluszeichen (++) symbolisiert. Eine teilweise Erfüllung wird durch ein einfaches Plus (+) dargestellt. Sollte die Anforderung nicht erfüllt sein, findet ein Minuszeichen (-) Verwendung. Dies dient der weiteren Strukturierung der SLR, und hilft, die Eignung der Quellen besser einzuordnen.

3.4 Synthese der Anwendungsmöglichkeiten

Im folgenden Kapitel werden die Ergebnisse der systematischen Literaturrecherche präsentiert, die durchgeführt wird, um Anwendungsmöglichkeiten und deren Umsetzungen von Natural Language Processing im Bereich des Application Lifecycle Management zu identifizieren und zu analysieren. Tabelle 3-4 zeigt das Suchprotokoll der Recherche.

Tabelle 3-4 Suchprotokoll der systematischen Literaturrecherche

Suchstring: ***"Natural language processing" AND ("Application Lifecycle Management" OR "Requirements Engineering" OR "Document* Management" OR "Test Management" OR "Bug and Issue Management")***

Datenbank	Erfassungsbereich	Anzahl der Suchergebnisse	Anzahl der Suchergebnisse nach Sichtung des			
			Titels	Abstracts	Inhalts	Ohne Duplikate
Scopus	2023-2024	172	134	73	21	21
IEEExplore	2023-2024	933	88	48	30	14
Web of Science	2023-2024	43	25	19	4	0
	Summe	1148	247	140	55	35

Das Suchprotokoll in Tabelle 3-4 umfasst eine Untersuchung zu wissenschaftlichen Artikeln unter Verwendung des Suchstrings "Natural Language Processing" in Verbindung mit den Begriffen "Application Lifecycle Management", "Requirements Engineering", "Document* Management", "Test Management" und "Bug and Issue Management". Die Suchanfrage wurde in drei großen wissenschaftlichen Datenbanken durchgeführt: Scopus, IEEExplore und Web of Science.

Wie in der Tabelle zu sehen ist ergab diese Suche zunächst 1148 Suchergebnisse. Diese Suchergebnisse werden in jeder Stufe nach den Kriterien der Bewertungsmatrix (Tabelle 3-3) aus Kapitel 3.3.2 bewertet. In den Stufen, in denen die Titel und die Abstracts bewertet werden, wird zuerst nur nach „+" und „-" bewertet, da noch keine Aussagen über die weiteren Kriterien

getroffen werden können. Durch diese stufenweise Bewertung konnten 35 relevante Arbeiten identifiziert werden.

Die in den Arbeiten behandelten Use Cases lassen sich in mehrere thematische Gruppen unterteilen, die verschiedene Aspekte der Anforderungsanalyse und des Softwareentwicklungsprozesses adressieren. Diese sind in Abbildung 3-3 aufgeführt. Die Details der einzelnen Arbeiten werden zudem im Folgenden tabellarisch aufgeführt. Die Auswertung und Diskussion der Quellen wird in Kapitel 4 durchgeführt.

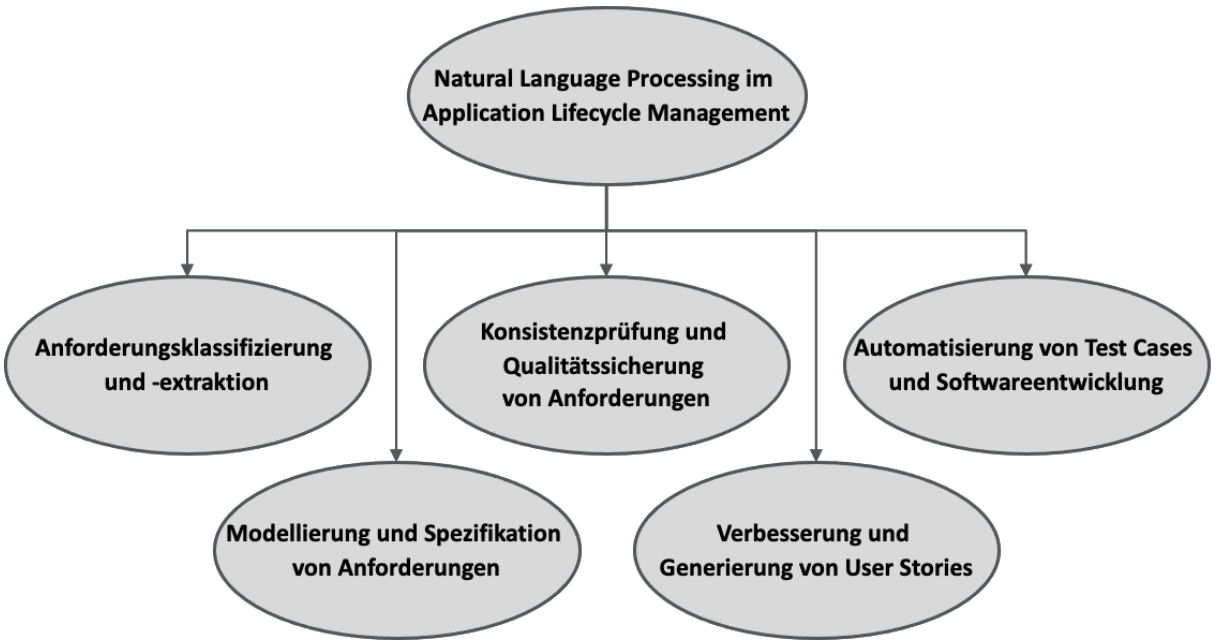


Abbildung 3-3 Thematische Gruppen der Use Cases aus der Literatur

Anforderungsklassifikation und -extraktion

Der Schwerpunkt dieser Gruppe liegt auf der Klassifizierung und Extraktion von Anforderungen, sei es zur Unterscheidung zwischen funktionalen und nicht-funktionalen Anforderungen, zur Extraktion von Modellen aus textuellen Anforderungen oder zur Klassifizierung von Fehlerberichten. NLP-Modelle wie BERT und GPT-Varianten bieten hierbei Möglichkeiten zur automatisierten Analyse großer Mengen von Anforderungen.

Tabelle 3-5 Ausgewertete Quellen in der Kategorie Anforderungsklassifikation und -extraktion

Quelle	Use Case	NLP Kategorie	Verwendete Sprachmodelle
Alhoshan et al. (2023)	Anforderungsklassifizierung mit Zero-Shot Learning	Textklassifizierung	S-BERT, All-MiniLM-L12 (AllMini)
Arulmohan et al. (2023)	Extraktion von Domänenmodellen aus textuellen Anforderungen	Informationsextraktion	GPT-3.5
Cruciani et al. (2023)	Klassifizierung von funktionalen und nicht-funktionalen Anforderungen	Textklassifizierung	BERT, DistilBERT, S-BERT
Das et al. (2023)	Named Entity Recognition in Software-Spezifikationen	Informationsextraktion	GPT-3, T5, RoBERTa

El-Hajjami et al. (2023)	Klassifizierung von funktionalen und nicht-funktionalen Anforderungen	Textklassifizierung	GPT-3, GPT-4
Gräßler et al. (2023)	Extraktion von Anforderungen im Automobilbau	Informationsextraktion, Textklassifizierung	BERT
Heo et al. (2024)	Klassifizierung von Fehlerberichten	Textklassifizierung	BERT, RoBERTa, GPT-3
Nasiri und Lahmer (2024)	Gruppierung von Agile User Stories zur Identifizierung von Duplikaten und Verbesserung der Backlog-Verwaltung	Informationsextraktion, Textklassifizierung	S-BERT
Rahman et al. (2023)	Klassifizierung von funktionalen und nicht-funktionalen Anforderungen	Textklassifizierung	BERT, GPT-2
Wang et al. (2024)	Automatisierte Abgrenzung von Anforderungen	Informationsextraktion, Textklassifizierung	DeBERTa, LLaMA 2, RoBERTa
Zhang et al. (2023a)	Bewertung der Leistung von ChatGPT beim Abrufen von Anforderungsinformationen in einer Zero-Shot-Umgebung	Informationsextraktion, Zusammenfassung	ChatGPT

Tabelle 3-5 gibt einen Überblick über die verschiedenen Anwendungsfälle, die NLP-Kategorien sowie die in den jeweiligen Studien verwendeten Sprachmodelle. BERT und seine Varianten (wie S-BERT und Distil-BERT) kamen häufig bei der Klassifizierung von Anforderungen zum Einsatz, während GPT-Modelle sowohl bei der Klassifikation als auch bei der Informationsextraktion verwendet wurden. Auch andere Modelle wie RoBERTa und T5 fanden mehrmals Anwendung.

Konsistenzprüfung und Qualitätssicherung von Anforderungen

Diese Gruppe befasst sich mit der Sicherstellung der Konsistenz und Qualität von Anforderungen. Hierzu gehören Überprüfungen der Konsistenz von Anforderungen, das Erkennen von Mehrdeutigkeiten und die Überprüfung von Anforderungen auf Vollständigkeit und Klarheit. Diese Ansätze nutzen NLP, um die Qualitätssicherung in der frühen Phase des Softwareentwicklungsprozesses zu unterstützen.

Tabelle 3-6 Ausgewertete Quellen in der Kategorie Konsistenzprüfung und Qualitätssicherung von Anforderungen

Quelle	Use Case	NLP Kategorie	Verwendete Sprachmodelle
Bertram et al. (2023)	Konsistenzprüfung von Automobilanforderungen	Informationsextraktion, Textklassifizierung	GPT-J-6B
Couder et al. (2024)	Verifizierung von Anforderungen durch Analyse des Quellcodes	Informationsextraktion, Textklassifizierung	GPT-3.5
Ezzini et al. (2023)	Unterstützung bei der Beantwortung von Fragen zur Analyse von Anforderungen in natürlicher Sprache	Informationsextraktion, Generierung natürlicher Sprache	BERT, t5

Gärtner et al. (2023)	Erkennung von Bedingungen innerhalb der Anforderungen	Informationsextraktion, Textklassifizierung	GPT-3, GPT-3.5, GPT-4, LlaMA
Jafari et al. (2023)	Auflösung von anaphorischer Mehrdeutigkeit in Software-Anforderungstexten	Informationsextraktion	BERT, RoBERTa, GPT-3.5, GPT-4, T5
Leong und Barbosa (2023)	Übersetzen von Anforderungen in natürlicher Sprache in formale Spezifikationen	Informationsextraktion, Generierung natürlicher Sprache	ChatGPT
Luitel et al. (2024)	Verbesserung der Vollständigkeit der Anforderungen durch automatisierte Unterstützung	Informationsextraktion, Generierung natürlicher Sprache	GPT
Zhang et al. (2023b)	Erkennung von Fehlern in Software-Anforderungstexten, insbesondere die Identifizierung von Problemen wie unscharfen, nicht überprüfbaren und unvollständigen Anforderungen	Informationsextraktion, Textklassifizierung	BERT

Tabelle 3-6 fasst die analysierten Quellen zur Konsistenzprüfung und Qualitätssicherung von Anforderungen zusammen. Die Tabelle zeigt die verschiedenen Anwendungsfälle, in denen NLP-Techniken zur Verbesserung der Konsistenz und Qualität von Anforderungstexten eingesetzt wurden. Sie bietet einen Überblick über die NLP-Kategorien, die in diesen Studien verwendet wurden, wie etwa die Informationsextraktion und Textklassifizierung, sowie die eingesetzten Sprachmodelle. Diese umfassen Modelle wie BERT, GPT (einschließlich GPT-3.5 und GPT-4), sowie ChatGPT und T5, die zur Analyse, Verifikation und Verbesserung der Anforderungstexte in verschiedenen Kontexten eingesetzt wurden.

Automatisierung von Test Cases und Softwareentwicklung

Diese Use Cases fokussieren sich auf die Automatisierung von Test Cases und Codegenerierung. Durch den Einsatz von NLP können aus textbasierten Anforderungen automatisch Test Cases und Code generiert werden. Dies erhöht die Effizienz und Zuverlässigkeit im Entwicklungsprozess.

Tabelle 3-7 Ausgewertete Quellen der Kategorie Automatisierung von Test Cases und Softwareentwicklung

Quelle	Use Case	NLP Kategorie	Verwendete Sprachmodelle
Boukhlif et al. (2024)	Überblick über NLP im Softwaretest, einschließlich Test Caseerstellung	Informationsextraktion, Textklassifizierung	BERT
Fischbach et al. (2023)	Automatische Generierung von Akzeptanztests aus den Anforderungen	Informationsextraktion	BERT, RoBERTa
Frattini et al. (2023)	Automatisierte Generierung von Test Casebeschreibungen aus natürlichsprachlichen Anforderungen	Informationsextraktion, Textklassifizierung	BERT, RoBERTa

Li et al. (2024)	Einsatz von ChatGPT zur Quellcodegenerierung und -verfeinerung unter Verwendung von Prompt-Engineering-Techniken	Informationsextraktion, Generierung natürlicher Sprache	ChatGPT
Mathur et al. (2023)	Automatisierte Generierung von Test Cases	Informationsextraktion, Generierung natürlicher Sprache	T5, GPT-3
Sawada et al. (2023)	Automatisierung der Zuordnung von Softwareanforderungen zu Test Cases zur Verbesserung der Softwarezuverlässigkeit und Verkürzung der Entwicklungszeit	Informationsextraktion, Textklassifizierung	BERT, GPT-3

Tabelle 3-7 gibt einen Überblick über die ausgewerteten Quellen zur automatisierten Generierung von Test Cases und Softwareentwicklung. In dieser Kategorie wird die automatische Erstellung von Test Cases sowie die Zuordnung von Anforderungen zu Tests untersucht. Die Tabelle zeigt, welche NLP-Kategorien und Sprachmodelle in den jeweiligen Studien verwendet wurden. Hierbei kommen häufig Informationsextraktion und Textklassifizierung zum Einsatz, wobei Modelle wie BERT, RoBERTa, GPT-3 und T5 sowie ChatGPT vertreten sind. Diese Modelle werden genutzt, um Test Case Beschreibungen zu generieren, Anforderungen zu analysieren und den Softwareentwicklungsprozess durch Automatisierung zu unterstützen.

Verbesserung und Generierung von User Stories

In dieser Gruppe liegt der Schwerpunkt auf der Verbesserung und Generierung von Anforderungen und User Stories. Insbesondere in agilen Entwicklungsprozessen bietet NLP Unterstützung bei der automatisierten Erstellung, Verfeinerung und Verwaltung von User Stories und Anforderungen.

Tabelle 3-8 Ausgewertete Quellen der Kategorie Verbesserung und Generierung von User Stories

Quelle	Use Case	NLP Kategorie	Verwendete Sprachmodelle
Dos Santos et al. (2024)	Automatische Generierung von User Stories in der agilen Entwicklung	Generierung natürlicher Sprache	GPT-3, N-gram model
Oswal et al. (2024)	Umwandlung von Softwareanforderungen in User Stories	Informationsextraktion, Generierung natürlicher Sprache	GPT-3.5
Scoggin und Torres Marques-Neto (2024)	Identifizierung und Validierung von User Stories	Textklassifizierung	BERT
Vito et al. (2023)	Verbesserung der Qualität von Anwendungsfällen	Informationsextraktion, Generierung natürlicher Sprache	ChatGPT

Tabelle 3-8 fasst die ausgewerteten Quellen zur Verbesserung und Generierung von User Stories zusammen. Die Tabelle gibt einen Überblick über verschiedene Anwendungsfälle, bei denen NLP-Techniken zur Generierung und Validierung von User Stories in der agilen Entwicklung eingesetzt wurden. Dabei kommen sowohl die Informationsextraktion als auch die Generierung natürlicher Sprache zur Anwendung. Zu den verwendeten Sprachmodellen zählen GPT-3, GPT-3.5 und ChatGPT, die genutzt werden, um User Stories automatisch zu

generieren oder ihre Qualität zu verbessern. Zudem findet in einer Studien die Textklassifizierung mit Modellen wie BERT Anwendung.

Modellierung und Spezifikation von Anforderungen

Diese Use Cases konzentrieren sich auf die Modellierung und Spezifikation von Anforderungen, insbesondere im Bereich eingebetteter Systeme und Problemrahmen. NLP unterstützt hierbei die automatische Erstellung und Verfeinerung von Anforderungsmodellen, die für die Systementwicklung entscheidend sind.

Tabelle 3-9 Ausgewertete Quellen der Kategorie Modellierung und Spezifikation von Anforderungen

Quelle	Use Case	NLP Kategorie	Verwendete Sprachmodelle
Ronanki et al. (2023)	Untersuchung des Potenzials von ChatGPT zur Unterstützung bei der Anforderungserhebung durch Vergleich der Ergebnisse mit menschlichen Experten	Informationsextraktion, Generierung natürlicher Sprache	ChatGPT
Ruan et al. (2023)	Automatisierte Generierung von Anforderungsmodellen für eingebettete Systeme mit ChatGPT und vordefinierten Kompositionsregeln	Informationsextraktion, Generierung natürlicher Sprache	ChatGPT
Seidel und Späthe (2024)	Entwicklung und Validierung von NLP-Algorithmen für Chatbots im Requirements Engineering	Informationsextraktion, Generierung natürlicher Sprache	GPT-2, BERT
Uygun und Momodu (2024)	Vereinfachung komplexer Anforderungsdokumente in der Automobilbranche	Informationsextraktion, Zusammenfassung	GPT
Wei et al. (2023)	Modellierung von Anforderungen mit NLP im Kontext von Problemrahmen	Informationsextraktion, Generierung natürlicher Sprache	BERT
Yeow et al. (2024)	Automatisierung von Aufgaben des Software Requirements Engineering	Informationsextraktion, Generierung natürlicher Sprache	GPT-3.5

Tabelle 3-9 gibt einen Überblick über die analysierten Quellen zur Modellierung und Spezifikation von Anforderungen. In dieser Kategorie wird untersucht, wie NLP-Techniken zur automatisierten Generierung und Vereinfachung von Anforderungsmodellen und -dokumenten verwendet werden. Die Tabelle zeigt, dass in vielen Studien Informationsextraktion und die Generierung natürlicher Sprache zum Einsatz kommen. Die genutzten Sprachmodelle in diesem Kontext sind ChatGPT, GPT-2, GPT-3.5 und BERT. Diese Modelle unterstützen die Automatisierung und Vereinfachung von Anforderungen, insbesondere in Bereichen wie eingebetteten Systemen und dem Software Anforderungsmanagement.

Im folgenden Abschnitt werden die Daten quantitativ ausgewertet, um einen Überblick der NLP und ALM Anwendungen zu bekommen.

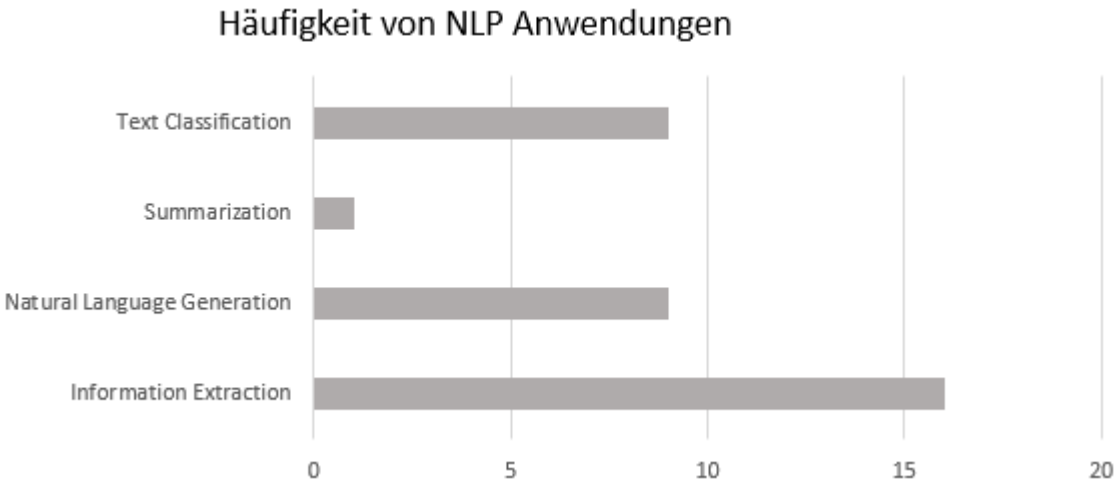


Abbildung 3-4 Häufigkeit der Verwendung von Natural Language Processing Anwendungen

In Abbildung 3-4 ist eine Übersicht der verwendeten NLP Anwendungen in den ausgewerteten Quellen zu sehen. Die Ergebnisse zeigen eine klare Präferenz für Information Extraction als primäre NLP-Anwendung, die in 16 Fällen identifiziert wurde. Dies deutet darauf hin, dass Information Extraction als besonders wertvoll für die Extraktion relevanter Informationen aus großen Textmengen angesehen wird und eine zentrale Rolle in der Verwaltung und Analyse von Informationen im ALM-Prozess spielt. Die häufige Nutzung von Text Classification und Natural Language Generation, jeweils in 9 Fällen, unterstreicht die Bedeutung dieser Methoden für die automatisierte Kategorisierung von Dokumenten sowie die Generierung von Texten zur Unterstützung von Entscheidungsprozessen. Die vergleichsweise seltene Anwendung von Summarization, die nur einmal vorkam, könnte darauf hindeuten, dass diese Methode entweder weniger relevant für die spezifischen Anforderungen im ALM ist oder dass sie durch andere Methoden ersetzt wird, die in der Lage sind, umfassendere oder spezifischere Informationen zu liefern. Insgesamt reflektieren diese Ergebnisse die unterschiedlichen Schwerpunkte und Prioritäten in der Anwendung von NLP-Techniken im Kontext des ALM.

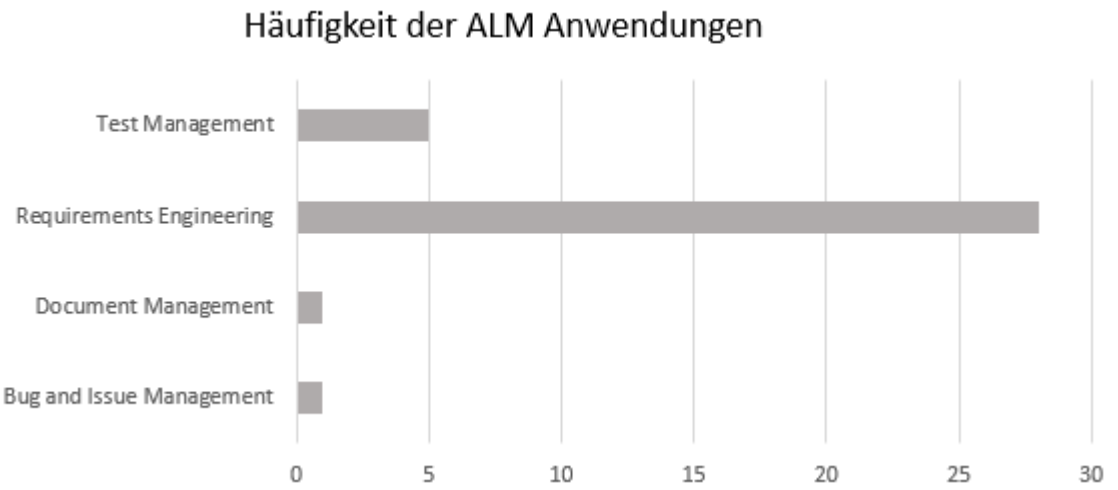


Abbildung 3-5 Häufigkeit der Verwendung von Application Lifecycle Management Anwendungen

In Abbildung 3-5 ist zusätzlich die Übersicht zu den ALM Anwendungen, für die die Use Cases entwickelt wurden. Die Analyse verdeutlicht eine deutliche Konzentration auf Requirements Engineering, das in 28 Fällen als Anwendung identifiziert wurde. Dies weist darauf hin, dass der Einsatz von NLP im Requirements Engineering besonders relevant ist, möglicherweise aufgrund der Komplexität und der zentralen Rolle, die das Anforderungsmanagement im

gesamten Lebenszyklus eines Softwareprojekts spielt. Die relativ häufige Verwendung von NLP im Test Management, das in 5 Fällen vorkommt, spiegelt die Notwendigkeit wider, Testprozesse effizient zu verwalten und zu optimieren, insbesondere durch die Automatisierung von Test Case Generierung und -klassifikation. Die seltenere Anwendung von NLP im Bug and Issue Management sowie im Document Management, jeweils nur einmal identifiziert, könnte darauf hinweisen, dass diese Bereiche entweder weniger stark von NLP profitieren oder dass alternative Ansätze zur Problemlösung bevorzugt werden. Insgesamt zeigen diese Ergebnisse, dass die Integration von NLP im ALM stark auf spezifische, kritische Phasen wie das Requirements Engineering fokussiert ist, während andere Bereiche bisher weniger durch NLP-Techniken unterstützt werden.

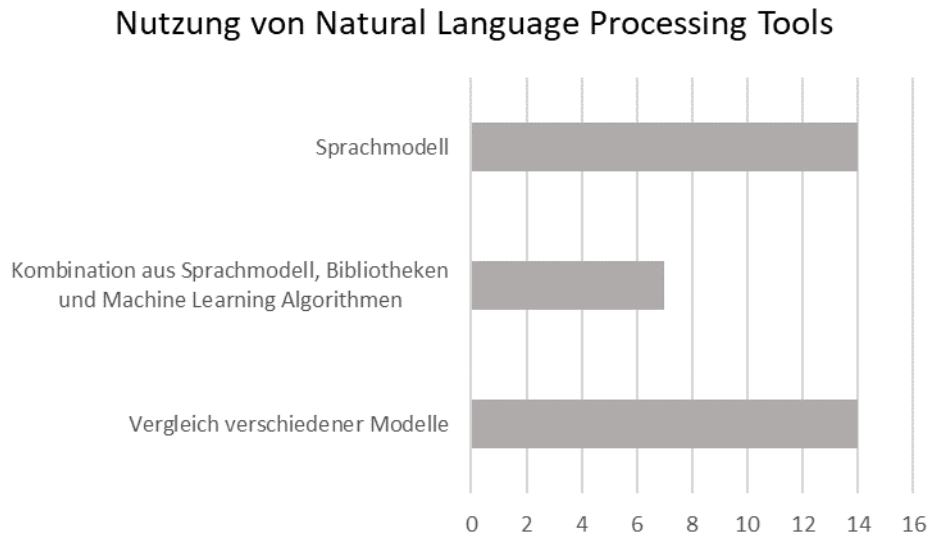


Abbildung 3-6 Verwendete Natural Language Processing Tools

In Abbildung 3-6 ist eine Übersicht Nutzung der NLP-Tools aufgeführt, die in der ausgewerteten Literatur verwendet wurden. Die gleichmäßige Verteilung zwischen den 14 Fällen, in denen ausschließlich Sprachmodelle verwendet wurden, und den 14 Fällen, in denen verschiedene Sprachmodelle verglichen wurden, zeigt die hohe Bedeutung der Auswahl und Feinabstimmung von Sprachmodellen für spezifische ALM-Anwendungen. Besonders interessant ist die Beobachtung, dass in 7 Fällen eine Kombination aus Sprachmodellen, NLP-Bibliotheken (häufig für Python) und Machine Learning-Algorithmen eingesetzt wurde, wobei diese Kombination hauptsächlich zur Vorverarbeitung der Daten und zur abschließenden Qualitätsklassifizierung diente. Dies deutet darauf hin, dass komplexere Anwendungsfälle im ALM eine hybride Herangehensweise erfordern, die die Stärken verschiedener Technologien kombiniert. Es ist zudem anzumerken, dass in den meisten Fällen eine Vorverarbeitung der Daten stattfand, diese jedoch entweder manuell durchgeführt wurde, oder durch andere, nicht weiter spezifizierte Tools durchgeführt wurde, die an dieser Stelle nicht ausgewertet werden konnten.

4 Entwicklung einer Methodik zur Einbindung von Natural Language Processing in Application Lifecycle Management

In diesem Kapitel wird eine Methodik zur Integration von Natural Language Processing in das Application Lifecycle Management entwickelt. Der zunehmende Einsatz von NLP in verschiedenen Bereichen der Softwareentwicklung eröffnet neue Möglichkeiten zur Optimierung von Prozessen und zur Effizienzsteigerung. Ziel dieses Kapitels ist es, die Anwendungsmöglichkeiten von NLP im Kontext von ALM zu analysieren, einen geeigneten Use Case auszuwählen, das Vorgehen zur exemplarischen Integration zu skizzieren und ein Validierungskonzept zu entwerfen. Zunächst werden die Ergebnisse der systematischen Literaturrecherche, dessen Methodik in Kapitel 3 erläutert wird analysiert und diskutiert. Die Analyse und Diskussion der Anwendungsmöglichkeiten beleuchten die Potenziale und Grenzen von NLP in ALM. Basierend auf dieser Analyse wird ein spezifischer Use Case ausgewählt, um die Umsetzbarkeit der Integration zu demonstrieren. Im Anschluss daran wird ein detailliertes Vorgehensmodell nach der CRISP-DM Methodik, die in Kapitel 2.1 behandelt wird, für die exemplarische Integration von NLP in ALM beschrieben. Abschließend wird ein Konzept zur Validierung der entwickelten Methodik vorgestellt, welches die Grundlage für eine fundierte Bewertung der Ergebnisse bildet.

4.1 Analyse der identifizierten Anwendungsmöglichkeiten zur Methodikentwicklung

Für die Entwicklung eines eigenen Konzepts zur Umsetzung eines Use Cases ist es notwendig, bestehende Lösungen zu analysieren, um Erkenntnisse aus früheren Forschungsergebnissen und deren Umsetzungen zu gewinnen. Im Folgenden werden für die einzelnen Kategorien, die in Kapitel 3.4 festgelegt werden, die spezifischen Use Cases und deren Umsetzung bewertet. Diese Ergebnisse dienen zur Beantwortung der Forschungsfrage F1 „Wie kann Natural Language Processing wertschöpfend für Application Lifecycle Management Anwendungen genutzt werden?“.

4.1.1 Anforderungsklassifikation und -extraktion

Die Ansätze zur Klassifikation und Extraktion von Anforderungen basieren größtenteils auf Zero-Shot- oder Low-Shot-Learning. Alhoshan et al. (2023) erzielten bei der Anforderungsklassifizierung mit Zero-Shot-Learning, unter Einsatz der Modelle BERT und DistilBERT, überzeugende Resultate. Ohne eine spezifische Feinabstimmung für das jeweilige Anwendungsgebiet besteht die Gefahr, weniger relevante Merkmale für die Klassifizierung heranzuziehen. Auch wenn die Ergebnisse im Allgemeinen positiv ausfielen, könnten Anforderungen, die vom üblichen Sprachgebrauch abweichen, nicht optimal erfasst werden. Transformerbasierte Modelle wie BERT bieten große Leistungsfähigkeit, zeigen jedoch Einschränkungen, wenn sie ohne Anpassung an spezifische Kontexte eingesetzt werden (Devlin et al. 2018). Die positiven Ergebnisse könnten demnach eher auf eine allgemeine Formulierung der Anforderungen zurückzuführen sein als auf ein tiefgehendes Verständnis durch das Modell. Eine weitere Studie zur Anforderungsklassifikation identifizierte ein kleines, optimiertes Modell als besonders effektiv im Few-Shot-Learning (El-Hajjami et al. 2023). Während GPT-4.0 zwar auf einer größeren Parameterbasis trainiert wurde, erzielte GPT-3.5-turbo in allen Kategorien eine präzisere Klassifizierung. Interessant ist hierbei die Beobachtung, wonach GPT-3.5-turbo bei Zero-Shot-Learning schlechter abschnitt als GPT-4.0, wodurch die insgesamt bessere Leistung größerer Modelle ohne Feinabstimmung deutlich wird. In der Arbeit von Wang et al. (2024) erwiesen sich hingegen die Few-Shot-Methoden bei Klassifizierungsaufgaben als weniger erfolgreich.

Heo et al. (2024) analysierten die Klassifikation von Fehlerberichten und verglichen die Leistung verschiedener auf BERT basierender Modelle, die mit unterschiedlichen Datensätzen

nachtrainiert wurden. Hier schnitt das speziell auf Fehlerberichte angepasste Modell, trotz kleinem Datensatz, am besten ab. Dagegen kamen Gräßler et al. (2023) zu dem Schluss, spezialisierte Machine-Learning-Algorithmen Anforderungen klassifizieren besser als das LLM BERT, auch wenn beide mit identischen Daten trainiert wurden. Cruciani et al. (2023) testeten verschiedene vortrainierte und feinjustierte LLMs und fanden BERT und DistilBERT am besten für die Klassifikation geeignet. Ähnlich untersuchten Rahman et al. (2023) die Problematik des Overfittings. Dies tritt auf, wenn ein Modell zu stark auf Trainingsdaten abgestimmt ist und deshalb auf neuen, unbekannten Datensätzen ungenauer arbeitet. Sie schlagen vor, dies durch den Einsatz eines großen, vielfältig trainierten Modells zu vermeiden. Nasiri und Lahmer (2024) stellten fest, besser trainierte Modelle weisen bei Klassifizierungsaufgaben eine höhere Leistungsfähigkeit auf.

Für die Extraktion von Entitäten aus Softwaredokumenten lieferte das Zero-Shot-Verfahren in der Studie von Das et al. (2023) die besten Ergebnisse, wobei das untrainierte Modell durch seine hohe Leistungsfähigkeit hervorstach. Arulmohan et al. (2023) stellten darüber hinaus fest, speziell angepasste Methoden erzielen die besten Ergebnisse. Bei ihrer Studie zur Extraktion von Domänenmodellen schnitt ein spezialisiertes Tool am besten ab, gefolgt vom untrainierten LLM GPT-3.5 und schließlich von herkömmlichen Machine-Learning-Algorithmen mit SpaCy.

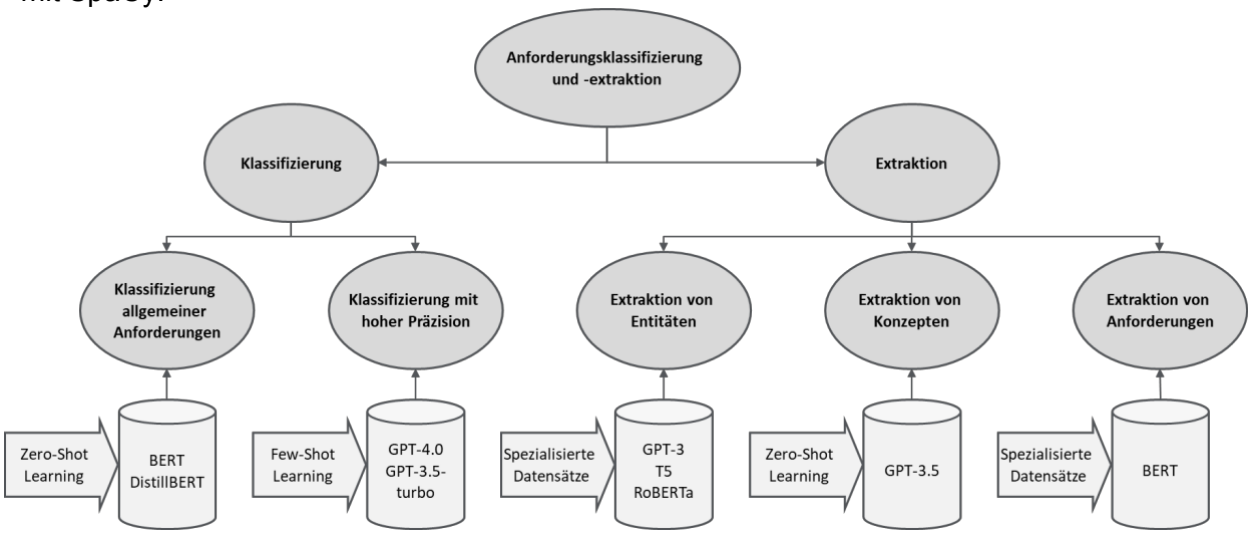


Abbildung 4-1 Übersicht der Artefakte und Umsetzungen für Anforderungsklassifizierung und -extraktion

Die Untersuchung von Zhang et al. (2023a) heben hervor, wie ChatGPT besonders in der Extraktion von Informationen zu Anforderungen eine hohe Recallrate aufweist, und damit eine große Anzahl relevanter Anforderungen erkennen kann. Dies ist in Zero-Shot-Lernumgebungen von Vorteil, da das Modell ohne vorheriges Training viele Informationen abdecken kann. Allerdings leidet manchmal die Präzision, wodurch eine sorgfältige Nachbearbeitung erforderlich ist. Ronanki et al. (2023) stellten ebenfalls die Fähigkeit von ChatGPT heraus, Anforderungen zu generieren, die abstrakt, konsistent und gut verständlich sind. Dennoch bleibt die Umsetzbarkeit der generierten Anforderungen eine Herausforderung, da es dem Modell an domänenspezifischem Wissen fehlt. Hier wird die Kombination von automatisierter Generierung und menschlicher Validierung als eine effektive Methode angesehen, um die Qualität und Umsetzbarkeit der Ergebnisse sicherzustellen.

Insgesamt zeigt sich, vortrainierte Sprachmodelle wie BERT, DistilBERT und GPT stellen flexible Werkzeuge für die Anforderungsanalyse im ALM dar und eignen sich für verschiedenste Anwendungsszenarien. Zero-Shot- und Few-Shot-Learning bieten dabei Lösungen für Kontexte mit limitierten Trainingsdaten, während spezialisierte Modelle besonders präzise Ergebnisse in anspruchsvollen Aufgaben ermöglichen. Für komplexe Szenarien können Ensemble-Methoden die Leistungsfähigkeit weiter erhöhen, erfordern jedoch eine sorgfältige Abwägung, um unerwünschte Effekte zu vermeiden (Rahman et al. 2023; Wang et al. 2024).

Zusammengefasst verdeutlichen die Untersuchungen, dass vortrainierte Sprachmodelle vielseitige und leistungsstarke Werkzeuge für die Anforderungsanalyse bieten. Dabei werden in den untersuchten Arbeiten vor allem verschiedene GPT und BERT Modelle eingesetzt. Während Zero-Shot- und Few-Shot-Learning besonders in Szenarien mit eingeschränkten Trainingsdaten vielversprechend sind, bieten spezialisierte Modelle signifikante Vorteile bei Aufgaben mit hohen Genauigkeitsanforderungen. Die Modellwahl und der Ansatz hängen somit stark vom jeweiligen Anwendungsfall ab und sollten entsprechend bewertet werden, um die bestmöglichen Ergebnisse zu erzielen. Für die Entwicklung eines eigenen Use Cases kann dies bedeuten, dass bei unzureichender Datenlage auf die Methoden des Zero-Shot oder Few-Shot-Learning zurückgegriffen wird. Auch die hohe Abdeckungsrate in diesen Fällen sollte beachtet werden. Für Use Cases mit spezifischen Anforderungen könnte hingegen ein spezialisiertes Modell geeigneter sein. Beispielsweise können die hohe Recallrate von ChatGPT oder die Präzision von spezialisierten Machine-Learning-Modellen als Ausgangspunkt für die Auswahl eines Modells für einen solchen Use Case dienen.

4.1.2 Konsistenzprüfung und Qualitätssicherung von Anforderungen

Zhang et al. (2023b) untersuchten die Fähigkeit von KI, Fehler in Anforderungen zu erkennen, und verglichen das LLM BERT mit einem Convolutional Neural Network. BERT überzeugte dabei durch kürzere Trainingszeiten und präzisere Ergebnisse. Luitel et al. (2024) prüften die Vollständigkeit von Anforderungen ebenfalls mit BERT und erzielten, sogar ohne zusätzliches Training auf spezifischen Datensätzen, gute Resultate.

Jafari et al. (2023) analysierten die Leistung verschiedener Modelle zur Lösung anaphorischer Mehrdeutigkeiten in Softwareanforderungen. Hierbei wurden die Modelle gezielt auf drei angepassten Datensätzen trainiert. Die Ergebnisse belegen insbesondere die Effektivität von BERT-Varianten und dem RoBERTa-Modell zur Erkennung und Auflösung von anaphorische Mehrdeutigkeiten. In dieser Aufgabe schnitten QA-Modelle insgesamt besser ab als textgenerierende Modelle wie T5, da letztere häufig zu "Halluzinationen" neigen dazu, Antworten zu generieren, die nicht zum Kontext passen oder ihm widersprechen. Dies mindert die Genauigkeit in kritischen Anwendungsfällen.

Bertram et al. (2023) integrierten GPT-J-6B, ein Open-Source-Modell mit 6 Milliarden Parametern, in ihre Toolchain zur Konsistenzprüfung von Anforderungen. Das Modell erwies sich dank Few-Shot-Learning als effizient, da es mit minimalem Zusatztraining auf spezifische Aufgaben abgestimmt werden kann. In einer weiteren Studie analysierten Couder et al. (2024) ein GPT-basiertes Modell zur Verifikation von Softwareanforderungen durch Quellcode-Analyse. Die Ergebnisse verdeutlichen die Fähigkeit von GPT-3.5, erfolgreich einfache und komplexe Anforderungen zu interpretierten und die Übereinstimmung mit dem Code zu überprüfen. Besonders wertvoll war die Fähigkeit, unvollständige oder nicht erfüllte Anforderungen zu erkennen und Verbesserungsvorschläge zu machen – ein klarer Hinweis auf das Potenzial von GPT-3.5 für die Verifikation komplexer Softwareanforderungen.

Leong und Barbosa (2023) beleuchteten die Automatisierung der Erstellung formaler Spezifikationen zur Verbesserung der Software-Verifizierbarkeit. Die Studie zeigte eine bessere Verarbeitung natürlicher Sprache durch ChatGPT im Vergleich zu symbolischen Methoden. Schwierigkeiten traten jedoch beim Umgang mit dem Format der Java Modeling Language auf, welche zur Zeit der Untersuchung eine Einschränkung für Programmieraufgaben darstellte. Ein weiteres Tool, QAssist, wurde von Ezzini et al. (2023) entwickelt und ermöglichte Stakeholdern, Fragen zu Anforderungen zu stellen und sofortige Antworten zu erhalten. Das auf BERT und T5 basierende Frage-Antwort-System erzielte nach gezieltem Fine-Tuning exzellente Ergebnisse.

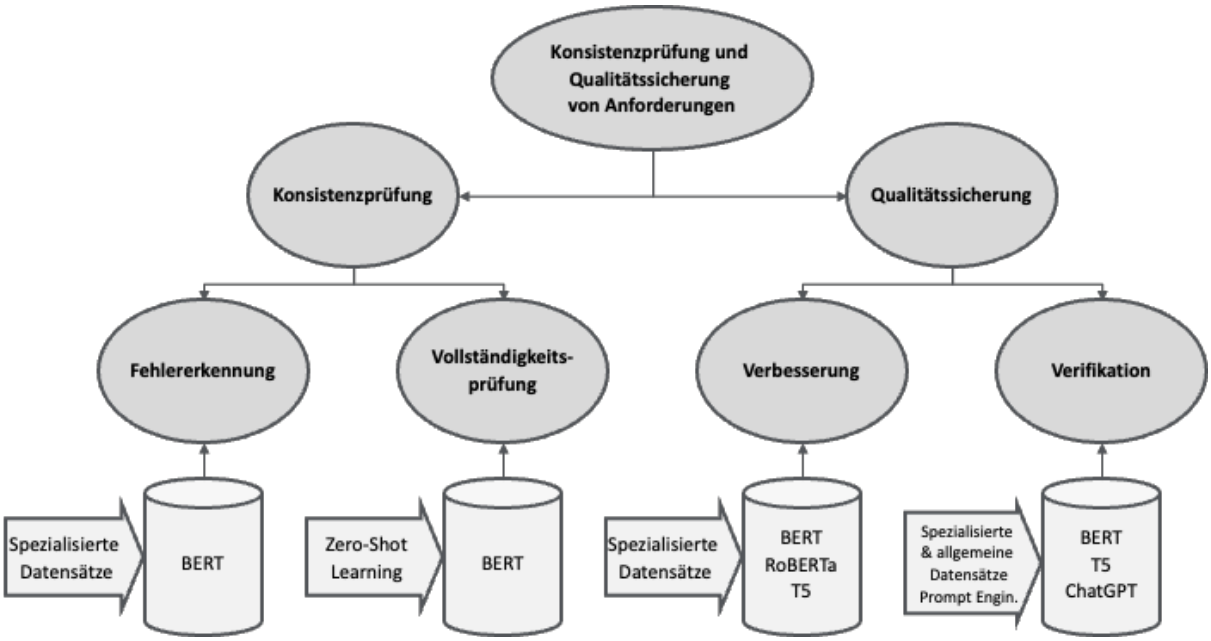


Abbildung 4-2 Übersicht der Artefakte und Umsetzungen für Konsistenzprüfung und Qualitätssicherung von Anforderungen

Zusammengefasst belegen diese Studien, LLMs wie BERT, RoBERTa, GPT-3.5 und GPT-J-6B leisten bereits wertvolle Dienste bei der Konsistenzprüfung und Fehlererkennung in Anforderungen. Sie bieten eine hohe Effizienz bei der Verifikation von Softwareanforderungen und erbringen oft schon ohne umfangreiches Training zuverlässige Ergebnisse. Dies unterstreicht ihre Flexibilität und den Nutzen in diversen Szenarien. Für einen eigenen Use Case könnte dies bedeuten, sich auf Modelle zu konzentrieren, die besonders für Aufgaben der Konsistenzprüfung geeignet sind, um eine solide Grundlage für die Methodik zu schaffen. Auch wenn der Use Case eine direkte Verbindung zwischen Anforderungen und Codequalität umfasst, könnten die genannten LLMs sinnvoll einsatzbar sein. Herausforderungen bestehen jedoch weiterhin, insbesondere hinsichtlich der Neigung zu Halluzinationen und der Schwierigkeiten in der Anwendung spezifischer Programmiersprachen. Insgesamt wird deutlich, die Kombination von NLP und LLMs besitzt das Potenzial die Qualitätssicherung im ALM entscheidend zu verbessern und den Softwareentwicklungsprozess effizienter zu gestalten. Es wird zudem deutlich, dass ein Fine-Tuning auf spezielle Datensätze für anspruchsvolle Aufgaben erforderlich ist.

4.1.3 Automatisierung von Test Cases und Softwareentwicklung

Die Automatisierung von Test Cases ist ein vielversprechender Bereich in der Softwareentwicklung, der durch die Fortschritte im Bereich der NLP-Modelle wie BERT, RoBERTa, T5 und GPT-3 an Bedeutung gewinnt. Verschiedene Studien, die im Folgenden behandelt werden, zeigen die Fähigkeit dieser Modelle, aus natürlichsprachlichen Anforderungen Test Cases zu generieren und Anforderungen präzise zuzuordnen.

Ein gemeinsames Thema in mehreren Arbeiten ist die Herausforderung wie entscheidend die Qualität der Eingabedaten für die Leistung der Modelle ist. So wiesen sowohl Frattini et al. (2023) als auch Fischbach et al. (2023) auf die Beeinträchtigung der Genauigkeit der generierten Test Cases für unklare oder weniger strukturierte Anforderungen hin. Während Frattini et al. ein Tool basierend auf BERT und RoBERTa entwickelten und gute Ergebnisse bei strukturierten Anforderungen erzielten, stellten Fischbach et al. fest, wie die Präzision stark abnimmt, wenn die Anforderungen nicht eindeutig formuliert sind. Ähnliche Beobachtungen zur Abhängigkeit von der Qualität der Daten machten auch Sawada et al. (2023), die ein BERT-Modell einsetzten, um Softwareanforderungen automatisiert Test Cases zuzuordnen.

Trotz der positiven Resultate war eine manuelle Überprüfung durch Ingenieure notwendig, um die Korrektheit sicherzustellen. Dies betont die wertvolle Unterstützung, die diese Modelle bieten, zeigt aber auch das Fehlen der vollständigen Automatisierung, die für den industriellen Einsatz erforderlich wäre.

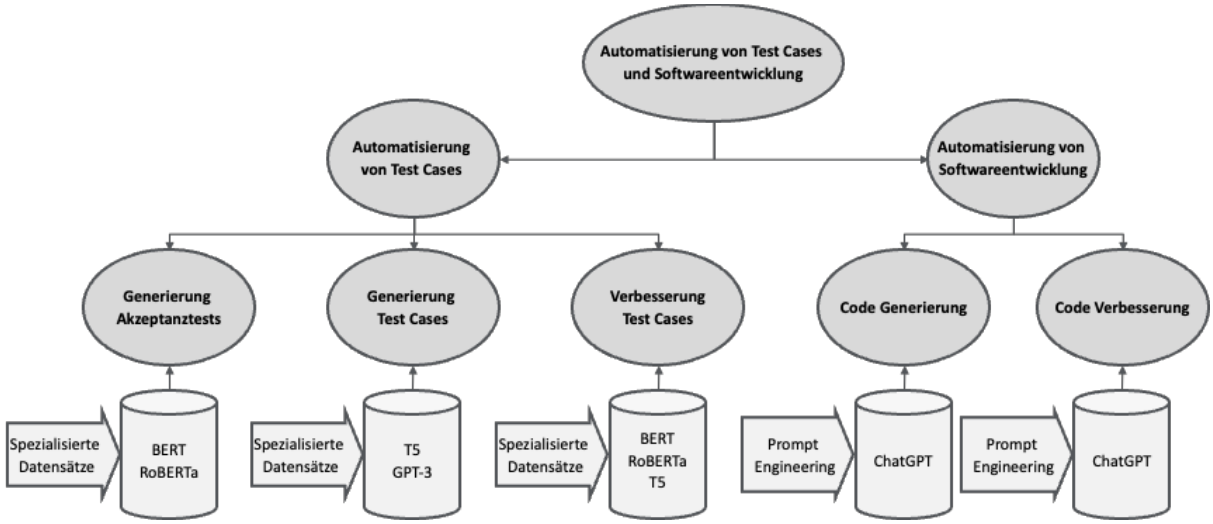


Abbildung 4-3 Übersicht der Artefakte und Umsetzungen für Automatisierung von Test Cases und Softwareentwicklung

Ein weiterer interessanter Ansatz ist die Kombination mehrerer Modelle zur Verbesserung der Test Case-Generierung. Mathur et al. (2023) kombinierten T5 und GPT-3, um aus Konversationen Test Cases zu erstellen, wobei T5 den Kontext extrahiert und GPT-3 detaillierte Zusammenfassungen verfeinert. Diese Kombination verdeutlicht, dass hybride Ansätze eine Möglichkeit bieten, die Komplexität natürlichsprachlicher Eingaben besser zu bewältigen. Auch hier zeigte sich jedoch, den großen Einfluss, den die Größe und Qualität der Trainingsdaten auf die Performance haben. Die Problematik von „Halluzinationen“, bei der Modelle fiktive oder fehlerhafte Informationen erzeugen, wurde von Li et al. (2024) in ihrer Untersuchung zur automatisierten Code-Generierung mit ChatGPT hervorgehoben. Diese Herausforderungen verdeutlichen, dass Optimierungen bei der Verarbeitung komplexer Anforderungen notwendig sind, um zuverlässige Ergebnisse zu erzielen.

Die Erkenntnisse aus diesen Studien verdeutlichen die Fortschritte in der Automatisierung von Test Cases und weiterhin Potenzial für praktische Anwendungen vorliegt, aber noch einige Hindernisse bestehen. Die Generalisierbarkeit der Modelle über verschiedene Anwendungsdomänen hinweg und die Fähigkeit, mit der Mehrdeutigkeit natürlicher Sprache umzugehen, bleiben zentrale Herausforderungen. Fortschritte in der Trainingsstrategie und die Verbesserung der Modellarchitektur könnten dazu beitragen, diese Lücken zu schließen und die Zuverlässigkeit der Systeme zu erhöhen. Hieraus wird ersichtlich, eine vollständige Automatisierung der Test Case-Generierung ist bisher nicht erreicht, bietet jedoch großes Potenzial und rückt mit der Weiterentwicklung von LLMs zunehmend in greifbare Nähe.

4.1.4 Verbesserung und Generierung von User Stories

Die Generierung und Verbesserung von User Stories und UML(Unified Modeling Language)-Use-Cases ist ein bedeutender Anwendungsbereich für NLP-Modelle, der in der agilen Softwareentwicklung zunehmend an Relevanz gewinnt. Verschiedene Studien, die im folgenden diskutiert werden, beleuchten, wie unterschiedliche Modelle und Ansätze eingesetzt werden, um die Qualität und Effizienz dieser Prozesse zu steigern.

Ein zentraler Aspekt ist die automatische Generierung von User Stories, bei der Dos Santos et al. (2024) den Vergleich zwischen einem N-Gramm-Modell und einem GPT-basierten Modell untersuchten. Die Ergebnisse zeigten die Überlegenheit von GPT in den meisten Metriken, jedoch Schwächen in der Semantik aufwies. Dies verdeutlicht die bestehenden Schwächen, selbst fortgeschrittener Modelle wie GPT, die semantische Nuancen nicht immer vollständig erfassen können, und unterstreicht den Bedarf an weitere Optimierungen.

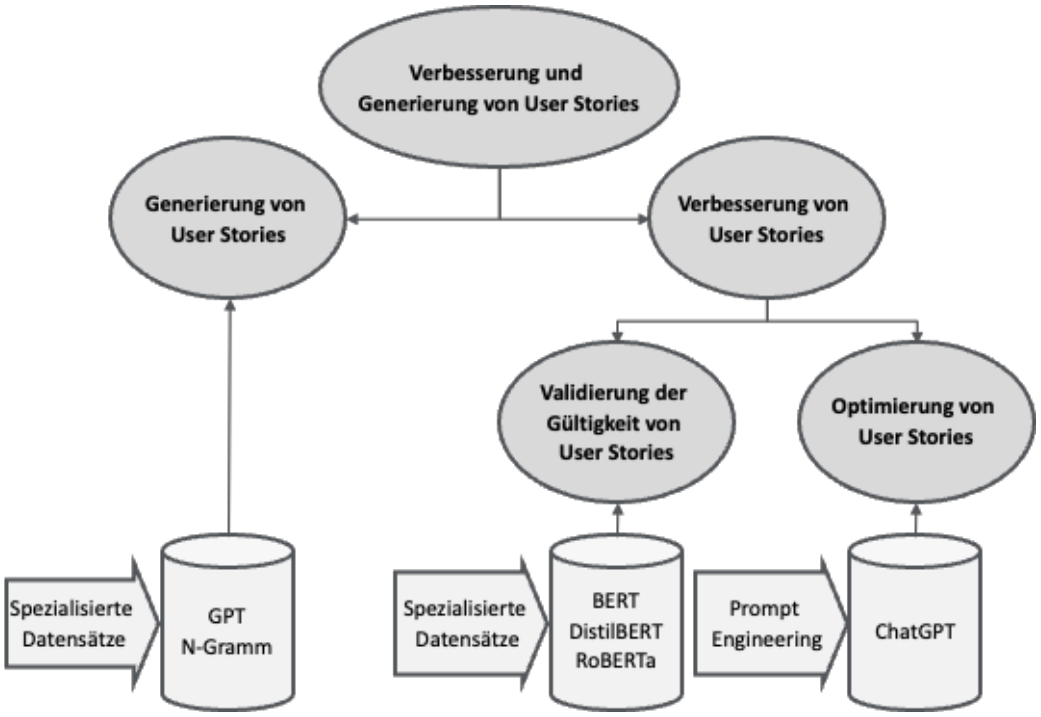


Abbildung 4-4 Übersicht der Artefakte und Umsetzungen für die Verbesserung und Generierung von User Stories

Die Verbesserung von UML-Use-Cases wurde von Vito et al. (2023) untersucht, die ein spezielles Tool auf Basis von ChatGPT entwickelten. Der Fokus lag hier stark auf dem „Prompt Engineering“, bei dem spezifische Prompts erstellt und kontinuierlich verfeinert wurden, um präzisere und hilfreichere Verbesserungsvorschläge zu erhalten. Die iterative Anpassung der Prompts basierte auf dem Feedback der Anwender und führte zu einer höheren Genauigkeit der Ergebnisse. Dies zeigt die Abhängigkeit des Erfolgs von NLP-Modellen, nicht nur von der Modellarchitektur, sondern auch von der Art und Weise, wie sie mit den richtigen Eingaben gesteuert werden.

Ein weiterer Aspekt der Verbesserung und Bewertung von User Stories ist die Validierung ihrer Gültigkeit. Scoggin und Torres Marques-Neto (2024) setzten BERT-base, DistilBERT-base und RoBERTa-base ein, um semantische Analysen von User Stories durchzuführen und ihre Validität zu bestimmen. Hierbei zeigte sich, RoBERTa erzielte aufgrund seiner umfangreichen Parametrisierung die besten Ergebnisse in Bezug auf Genauigkeit, Präzision, Rückrufrate und F1-Score. Dies verdeutlicht, eine umfassendere Modellarchitektur führt zu einer besseren Leistung bei anspruchsvollen Aufgaben.

Die Studien zeigen insgesamt die Effektivität von GPT-basierten Modellen für die Generierung qualitativ hochwertiger User Stories trotz ihrer semantischen Herausforderungen. Gleichzeitig verdeutlichen sie, gezieltes Prompt Engineering in Kombination mit Modellen wie ChatGPT unterstützt die Verbesserung von UML-Use-Cases. Zudem unterstreicht die Validierung von User Stories durch Modelle wie RoBERTa das Potential der Leistungssteigerung durch eine starke Parametrisierung und gezieltes Training. Diese Erkenntnisse betonen das Potenzial von NLP-Techniken, agile Softwareentwicklung effizienter zu gestalten und die Qualität der entwickelten Artefakte signifikant zu verbessern.

4.1.5 Modellierung und Spezifikation von Anforderungen

Die Modellierung von Anforderungen beschreibt den Prozess, Anforderungen in abstrahierte, strukturierte und visuell darstellbare Modelle zu überführen, wie beispielsweise UML-Diagramme. Ziel ist es, die Anforderungen verständlich zu machen, ihre Analyse und Validierung zu unterstützen und eine Grundlage für die Systementwicklung zu schaffen. Im Gegensatz dazu umfasst die Spezifikation von Anforderungen die präzise und textuelle Beschreibung aller Anforderungen, typischerweise in standardisierten Formaten wie Lasten- oder Pflichtenheften. Sie dient als verbindliche Basis für die Entwicklung und die Abstimmung zwischen Stakeholdern. In Kapitel 4.1.1 wurde zudem schon auf die Extraktion von Anforderungen eingegangen. Dieses Feld bezieht sich auf die Identifikation relevanter Anforderungsinformationen aus unstrukturierten oder semi-strukturierten Quellen, wie Dokumenten, E-Mails oder Interviews. Während die Extraktion oft den ersten Schritt darstellt, um Rohanforderungen zu gewinnen, fokussiert die Spezifikation auf deren schriftliche Ausarbeitung, und die Modellierung stellt diese Anforderungen schließlich in einer strukturierten, visuellen Form dar, um die Weiterverarbeitung zu erleichtern.

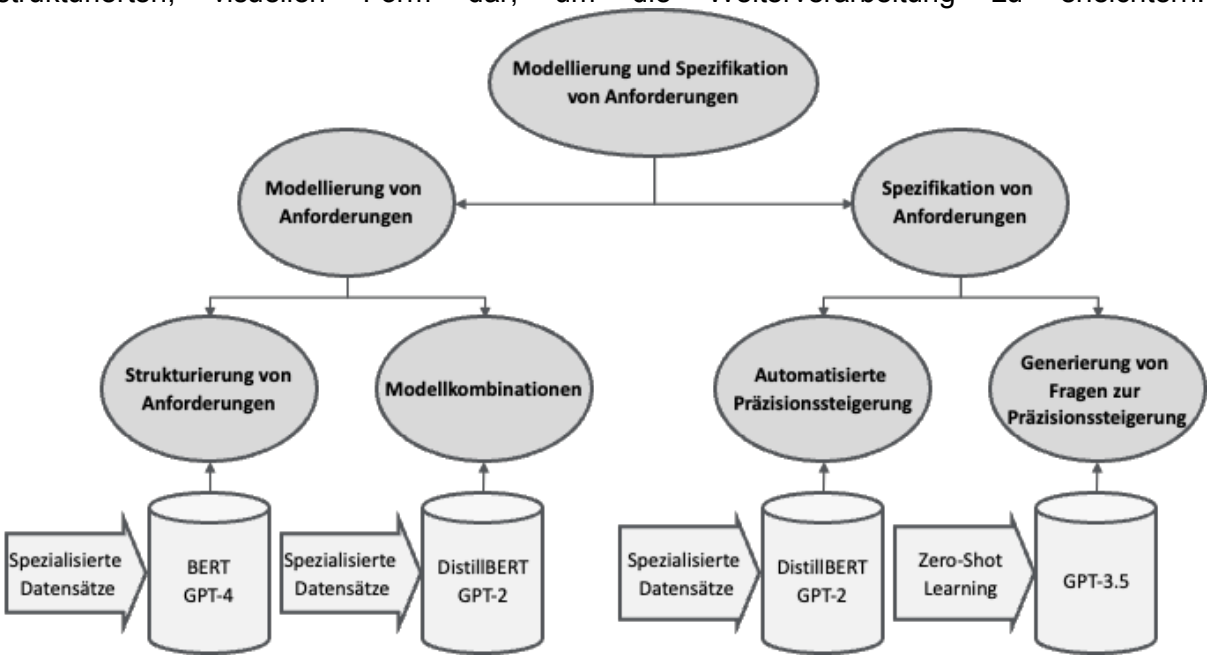


Abbildung 4-5 Übersicht der Artefakte und Umsetzungen für die Modellierung und Spezifikation von Anforderungen

Wei et al. (2023) zeigten die Fähigkeit von BERT, Schlüsselpunkte aus Anforderungstexten zu extrahieren, um den Modellierungsprozess zu unterstützen. Interessanterweise wurde das Modell ohne spezifisches Fine-Tuning verwendet und weist auf das Potenzial von BERT zur Verarbeitung allgemeiner Texte hin. Dennoch bleibt Raum für weitere Anpassungen, um die Ergebnisse zu verfeinern. Ruan et al. (2023) verfolgten einen anderen Ansatz und entwickelten ein automatisiertes Framework, das ChatGPT zur Extraktion und Strukturierung von Anforderungselementen aus natürlicher Sprache nutzt. Während das Framework für einfache Konstrukte effektiv war, traten bei komplexeren Anforderungen Einschränkungen auf. Diese Limitationen verdeutlichen den Bedarf an weiterentwickelten Modellen wie GPT-4, um die Leistung zu steigern. Ein weiterer Bereich, in dem NLP-Modelle Vorteile bieten, ist das Question Answering zu bestehenden Anforderungen. Modelle wie BERT können hier präzise Antworten liefern und die Konsistenz der Anforderungsdokumentation verbessern. Die Forschung legt die Erhöhung der Relevanz und Genauigkeit nahe, wenn generische LLMs an spezifische Domänen angepasst werden. Dies ist besonders wichtig, um sicherzustellen, dass die Ergebnisse den Anforderungen komplexer, branchenspezifischer Projekte gerecht werden. Die Automatisierung von Routinetätigkeiten mithilfe von NLP-Tools reduziert zudem den manuellen Aufwand und erhöht die Effizienz im gesamten Anwendungslebenszyklus. Dies unterstreicht, wie wichtig die Balance zwischen generativen Modellen und klassischen NLP-Techniken ist, um das Potenzial der Anforderungsanalyse voll auszuschöpfen.

Seidel und Späthe (2024) integrierten DistilBERT und GPT-2 in ihren Ansatz zur Anforderungsmodellierung, wobei Trainingsdaten verwendet wurden, die von GPT-4 generiert wurden. Dies zeigt, wie verschiedene Modelle und Trainingstechniken kombiniert werden können, um bessere Ergebnisse zu erzielen. Eine manuelle Überprüfung der Ergebnisse bestätigte die Genauigkeit der Modellvorhersagen. Einen besonderen Fokus auf die Erstellung präziser Fragen legten Yeow et al. (2024) mit der Implementierung von GPT-3.5. Das Modell generierte konsistente Fragen mit überdurchschnittlicher Klarheit und Relevanz, stieß jedoch auf Schwierigkeiten bei branchenspezifischen Inhalten, insbesondere im Umgang mit Fachbegriffen. Da hier kein weiteres Fine-Tuning stattfand, waren diese Herausforderungen erwartbar. Uygun und Momodu (2024) zeigten, wie gezieltes Fine-Tuning und der Einsatz spezialisierter Modelle, wie das Open-Source-LLM 'Nous-Hermes-13B-GPTQ', die Effizienz und Präzision bei der Dokumentenanalyse steigern können. Die lokale Implementierung des Modells trug dazu bei, Datenschutzanforderungen zu erfüllen und eine effiziente Verarbeitung komplexer, technischer Texte zu gewährleisten. Dies ist besonders relevant für die Automobilindustrie, wo hohe Anforderungen an die Genauigkeit und das Verständnis des Kontextes bestehen.

Die Analyse dieser Studien verdeutlicht die wichtige Rolle, die NLP-Modelle bei der Modellierung und Spezifikation von Anforderungen spielen können. Während generische Modelle wie BERT und GPT bereits ohne umfassendes Fine-Tuning gute Ergebnisse liefern, sind spezialisierte und feinabgestimmte Modelle wie 'Nous-Hermes-13B-GPTQ' besonders geeignet, um die Anforderungen in spezifischen Domänen zu erfüllen. Diese Beispiele zeigen, dass die Zukunft der NLP-gestützten Modellierung in der kontinuierlichen Weiterentwicklung und Anpassung der Modelle liegt, um deren Potenzial für komplexe Anwendungsfälle voll auszuschöpfen. Für die Umsetzung eines Use Cases können folgende Kriterien abgeleitet werden:

- Mächtige LLMs können für viele Themen bereits ohne spezifisches Fine-Tuning verwendet werden.
- Branchenspezifische Trainingsdaten bieten großes Potential zur Verbesserung der Ergebnisse.
- Eine Evaluierung der Ergebnisse durch Experten ist weiterhin nötig, um die Qualität abzusichern und falsche Ergebnisse zu verhindern.

4.2 Auswahl des Use Cases

Für die Auswahl eines geeigneten Use Cases zur Demonstration einer Demo-Anwendung ist es wichtig, einen Fall zu wählen, der nicht nur die Relevanz der Technologie aufzeigt, sondern auch die Leistungsfähigkeit und Grenzen moderner NLP-Modelle verdeutlicht. In Kapitel 4.1 werden verschiedene Anwendungsbereiche und deren Umsetzung detailliert untersucht, wobei deutlich wird, dass die Automatisierung der Test Case Generierung ein besonders vielversprechender Use Case ist. Dieser Use Case ermöglicht eine praxisnahe Demonstration der Wertschöpfungspotenziale von NLP-Technologien im Application Lifecycle Management.

Die Automatisierung der Test Case Generierung ist von besonderem Interesse, da sie einen der zeitintensivsten und fehleranfälligsten Prozesse in der Softwareentwicklung betrifft. Manuelle Test Case Erstellung erfordert erheblichen Aufwand und ist anfällig für Inkonsistenzen und unvollständige Testabdeckungen. Dies wirkt sich negativ auf die Qualität und Effizienz des gesamten Entwicklungsprozesses aus. Eine automatisierte Lösung kann diese Herausforderungen adressieren, indem sie die Erstellung von Test Cases beschleunigt und konsistentere Ergebnisse liefert. Dies führt zu einer verbesserten Testabdeckung und erleichtert die Nachverfolgbarkeit (Traceability), indem Anforderungen automatisch mit den entsprechenden Test Cases verknüpft werden. Ein weiterer Grund für die Wahl dieses Use Cases ist, dass er einen Vergleich aktueller Modelle ermöglicht. Mathur et al. (2023) haben in einer ähnlichen Untersuchung das Open-Source-LLM T5, sowie GPT-3 evaluiert. T5 wurde zusätzlich mit einem allgemeinen Datensatz trainiert. Die Ergebnisse fielen uneinheitlich aus,

da insbesondere T5 ohne spezialisierte Trainingsdaten nicht die gewünschte Genauigkeit erreichte.

In dieser Arbeit wird überprüft, ob T5 auch mit einem sehr kleinen, spezialisierten Datensatz akzeptable Ergebnisse liefern kann und inwiefern sich die Leistung von GPT in seiner neuesten Version, GPT-4o verbessert hat. Diese Analyse bietet wichtige Einblicke in die Eignung dieser Modelle für spezialisierte Anwendungsfälle und die Notwendigkeit von Anpassungen und Trainingsstrategien. Ein wesentlicher Aspekt bei der Integration solcher Technologien in Unternehmen ist der Schutz sensibler Daten. Um die Einhaltung von Datenschutzbestimmungen zu gewährleisten, müssen Lösungen gefunden werden, die eine sichere Verarbeitung von Daten ermöglichen. Hier bieten sich geschlossene Cloud-Lösungen wie Azure an, die die Vorteile leistungsfähiger NLP-Modelle mit den Anforderungen an Datenschutz und Datensicherheit vereinen. Die Wahl dieses Use Cases ist daher sinnvoll, um die Potenziale und Herausforderungen der Automatisierung von Test Cases durch NLP zu analysieren. Es ermöglicht, praxisrelevante Erkenntnisse zu gewinnen, die sowohl die Effizienzsteigerung als auch die Verbesserung der Qualität in der Softwareentwicklung betreffen, und bietet eine wertvolle Grundlage für die Beurteilung der Anwendbarkeit von NLP-Technologien im ALM.

4.3 Vorgehen zur exemplarischen Integration

Die exemplarische Integration von NLP in die Test Case Automatisierung kann strukturiert durch das CRISP-DM Vorgehensmodell erfolgen. Dieses Modell umfasst sechs Phasen: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation und Deployment. Im Folgenden werden diese Phasen detailliert beschrieben, um den Prozess der Test Case Automatisierung durch NLP vollständig zu erklären. Dieses Vorgehen bildet die Basis zur Bearbeitung der Forschungsfrage F2: „Wie kann die Integration von Natural Language Processing im Application Lifecycle Management durchgeführt werden?“.

4.3.1 Business Understanding

Das Projekt beginnt in der Phase des Business Understandings, in der das übergeordnete Ziel des Vorhabens klar definiert wird. Diese Phase wird in Kapitel 4.1 mit der Analyse der wissenschaftlichen Literatur zu diesem Thema bearbeitet. In Kapitel 4.2 wird daraus ein sinnvoller Use Case abgeleitet, für dessen Implementierung im Folgenden die Methodik hergeleitet wird. Zusammenfassend wird an dieser Stelle nochmal auf die wichtigsten Aspekte eingegangen, um die Hintergründe vollständig darzustellen.

Wie in Kapitel 2.3 erläutert, stellt das Requirements Engineering im Kontext des Application Lifecycle Managements eine zentrale Aufgabe dar. Diese Disziplin umfasst das Erheben, Dokumentieren, Verwalten und Nachverfolgen von Anforderungen, die oft in komplexen und umfangreichen Projekten auftreten. Dabei ist die Sicherstellung der Nachvollziehbarkeit zwischen Anforderungen und Test Cases essenziell, um eine konsistente und fehlerfreie Produkt- bzw. Anwendungsentwicklung zu gewährleisten. Ein wesentlicher Bestandteil des Requirements Engineering ist die Erstellung von Test Cases, die auf den definierten Anforderungen basieren. Diese Test Cases dienen dazu, die Funktionalitäten und Eigenschaften eines Produkts oder Systems zu überprüfen. Der Prozess der Test Case Generierung ist jedoch häufig zeitaufwendig und erfordert intensive manuelle Arbeit. Insbesondere in Projekten mit einer Vielzahl von Anforderungen steigt der Aufwand exponentiell an. Hinzu kommt die Anfälligkeit für Fehler bei der manuellen Erstellung von Test Cases, wie etwa inkonsistente oder unvollständige Tests, die letztlich zu Qualitätseinbußen führen können. Angesichts der Komplexität dieser Aufgabe sind Zero-Shot- und Few-Shot-Learning-Ansätze in Kombination mit Modellen mit einer geringen Parameteranzahl nicht geeignet. Da zudem keine ausreichende Menge an Trainingsdaten verfügbar ist, empfiehlt sich der Einsatz eines leistungsfähigen LLM. Wie in Abschnitt 2.4.3 erarbeitet, zählen derzeit

GPT-4o, Llama 3.1, Claude 3.5 und Google Gemini 1.5 zu den fortschrittlichsten Modellen auf dem Markt.

Das Hauptziel des Use Cases besteht daher darin, durch die Automatisierung der Test Case Generierung den manuellen Aufwand signifikant zu reduzieren und gleichzeitig die Qualität sowie die Konsistenz der Test Cases zu steigern. Mithilfe von NLP-Methoden soll ein Prozess entwickelt werden, der Anforderungen automatisch in passende Test Cases transformiert. Dies spart nicht nur Ressourcen, sondern ermöglicht es auch, schneller auf Änderungen in den Anforderungen zu reagieren und ist besonders in agilen Entwicklungsumgebungen von Vorteil.

Die spezifische Umsetzung des Use Cases des Projekts umfasst die Transformation von Anforderungen in Test Cases. Diese Aufgabe wird der Kategorie der Text-zu-Text-Generierung zugeordnet. Ziel ist es, aus vorliegenden Anforderungen direkt passende Test Cases abzuleiten, wobei die Nachverfolgbarkeit zwischen Anforderungen und den daraus resultierenden Test Cases gewährleistet wird. Die Automatisierung dieser Aufgabe bietet somit nicht nur eine Entlastung für Teams im Requirements Engineering, sondern schafft auch die Grundlage für eine effizientere und effektivere Qualitätssicherung.

4.3.2 Data Understanding

In der Phase des Data Understandings wird die Struktur der vorliegenden Daten im Detail analysiert, um ihre Eignung für die Modellierung und Evaluierung zu bewerten. Die bereitgestellte Datenbasis besteht aus Anforderungsdokumenten und den zugehörigen Test Cases im ReqIF-Format (Requirements Interchange Format). Dieses Format bietet eine XML-basierte Struktur, die es ermöglicht, Anforderungen und Test Cases in standardisierter und nachvollziehbarer Weise zu organisieren. Auf diese Datenstruktur wird in Kapitel 2 genauer eingegangen.

Tabelle 4-1 Übersicht der wichtigsten Tags und ihre Bedeutungen in ReqIF

Tag	Bedeutung
`<REQ-IF>`	Das Root-Element des ReqIF-Dokuments, das die gesamte Struktur enthält.
`<THE-HEADER>`	Enthält Metadaten über die Datei, wie den Ersteller, das Erstellungsdatum und Toolinformationen.
`<CORE-CONTENT>`	Der Hauptbereich des Dokuments, der die eigentlichen Anforderungen, Test Cases und deren Beziehungen enthält.
`<REQ-IF-CONTENT>`	Ein Unterbereich von `<CORE-CONTENT>`, in dem die wichtigsten Datenstrukturen wie Anforderungen und Spezifikationen organisiert sind.
`<SPECIFICATIONS>`	Organisiert die Objekte in einer Struktur, wie hierarchischen Listen oder Tabellen.
`<SPEC-OBEJCTS>`	Listet alle spezifischen Objekte (z. B. Anforderungen, Test Cases) auf, die in der Datei enthalten sind.
`<SPEC-OBEJCT>`	Repräsentiert ein einzelnes Objekt (Anforderung oder Test Case) mit einer eindeutigen Identifikation und zugehörigen Attributen.
`<DATATYPES>`	Definiert die verfügbaren Datentypen für Attribute, wie Text, numerische Werte oder Listen.

<SPEC-TYPES>	Beschreibt die verschiedenen Typen von Spezifikationen (z. B. Anforderung, Test Case) und ihre möglichen Attribute.
<VALUES>	Enthält die Werte der Attribute für ein spezifisches Objekt, wie den Text einer Anforderung oder zusätzliche Metadaten.
<ATTRIBUTE-VALUE-STRING>	Speichert einfache Text- oder String-Werte, wie IDs oder Namen.
<SPEC-RELATIONS>	Definiert Beziehungen zwischen Objekten, z. B. zwischen einer Anforderung und einem Test Case.

In Tabelle 4-1 sind wichtigsten Tags des ReqIF-Formats zusammengefasst, wenn Anforderungen extrahiert werden sollen. Zu den wichtigsten Elementen innerhalb der Datenstruktur zählen die sogenannte SPEC-OBJECTS, welche die einzelnen Anforderungen und Test Cases repräsentieren. Jedes dieser Objekte ist durch eine eindeutige Identifikationsnummer (IDENTIFIER) gekennzeichnet, die die Verknüpfung zwischen Anforderungen und Test Cases sowie deren Nachvollziehbarkeit sicherstellt. Ergänzt wird diese Struktur durch Metadaten, wie den Zeitpunkt der letzten Änderung (LAST-CHANGE), die eine Historisierung von Änderungen ermöglicht. Die Inhalte der Anforderungen und Test Cases sind in den Attributen der SPEC-OBJECTS gespeichert, insbesondere in den VALUES-Sektionen. Hier finden sich unterschiedliche Typen von Attributwerten, darunter ATTRIBUTE-VALUE-STRING, die einfache Textwerte oder IDs enthalten, sowie ATTRIBUTE-VALUE-XHTML, welche den eigentlichen Text der Anforderungen und Test Cases in XHTML-Formatierung bereitstellen. Diese Inhalte müssen extrahiert und vorverarbeitet werden, um überflüssiges Markup zu entfernen und den Text für die Modellierung nutzbar zu machen. Darüber hinaus spielen Beziehungen zwischen Anforderungen und Test Cases, die in SPEC-RELATIONS definiert sind, eine zentrale Rolle. Diese Beziehungen ermöglichen es, die Nachverfolgbarkeit zwischen Anforderungen und den daraus abgeleiteten Test Cases herzustellen und bieten damit eine Grundlage für eine konsistente Evaluation.

Ein besonderes Augenmerk liegt in dieser Phase darauf, die Struktur und den Inhalt der Daten so aufzubereiten, dass die Informationen aus der ReqIF-Datei effizient extrahiert und für die weitere Verarbeitung genutzt werden können. Die Konsistenz und Vollständigkeit der bereitgestellten Daten sind dabei entscheidend, um eine zuverlässige Weiterverarbeitung sicherzustellen. Insbesondere müssen die Beziehungen zwischen Anforderungen und Test Cases sowie die relevanten Attribute korrekt identifiziert und organisiert werden, um sie für die nächsten Schritte des Projekts nutzbar zu machen. Durch diese Analyse gewährleistet die Erfüllung der Anforderungen der Modellierung,alle relevanten Aspekte der Anforderungen und Test Cases korrekt abzubilden und für die Automatisierung nutzbar gemacht werden zu können.

4.3.3 Data Preparation

In der Phase der Data Preparation liegt der Fokus auf der Aufbereitung der Daten, um sie für die Verarbeitung durch NLP-Modelle nutzbar zu machen. Da die Daten im ReqIF-Format vorliegen, ist es erforderlich, sie zunächst zu parsen und die relevanten Inhalte in eine maschinenlesbare Struktur zu überführen. Der folgende Ansatz wird implementiert, um Anforderungen systematisch aus der komplexen XML-basierten Struktur zu extrahieren und in einem geeigneten Format abzuspeichern.

Zunächst werden die Daten mithilfe der Python-Bibliothek xml.etree.ElementTree geparst, um die hierarchische Struktur der ReqIF-Datei zu analysieren. Unter Verwendung der definierten XML-Namensräume wird der Abschnitt <CORE-CONTENT> lokalisiert, der die Hauptdaten enthält. Innerhalb dieses Abschnitts werden die Spezifikationstypen (<SPEC-TYPES>) identifiziert, die verschiedene Arten von Objekten in der Datei beschreiben, wie beispielsweise Anforderungen oder Test Cases. Für Anforderungen wird speziell nach dem Tag <SPEC-

OBJECT-TYPE> gesucht, dessen Attribut LONG-NAME den Begriff „Requirement“ enthält. Die entsprechenden Typen werden in einer Datenstruktur gespeichert, um sie in weiteren Verarbeitungsschritten zu referenzieren.

Tabelle 4-2 Verwendete Bibliotheken für die Datenvorverarbeitung

Bibliothek	Erklärung
xml.etree.ElementTree	Eine Bibliothek in Python für die Arbeit mit XML-Daten. Ermöglicht das Parsen, Navigieren und Bearbeiten von XML-Dateien.
csv	Eine Bibliothek in Python für die Arbeit mit CSV(Comma-seperated values)-Dateien. Ermöglicht das Lesen, Schreiben und Bearbeiten tabellarischer Daten im CSV-Format.

Im nächsten Schritt erfolgt die Extraktion der Anforderungen aus dem <SPEC-OBJECTS>-Abschnitt. Jedes <SPEC-OBJECT>repräsentiert eine Anforderung oder einen Test Case und enthält einen Verweis auf seinen Typ. Anhand der zuvor identifizierten Typen wird geprüft, ob das Objekt tatsächlich eine Anforderung ist. Anschließend werden die relevanten Inhalte aus den <ATTRIBUTE-VALUE-XHTML>-Tags extrahiert, die häufig die Titel und Beschreibungen der Anforderungen enthalten. Hierbei wird das XHTML-Markup bereinigt, um reinen Text zu gewinnen, der maschinenlesbar ist. Die extrahierten Anforderungen werden in einer Liste gespeichert, wobei jede Anforderung durch ihre ID, den zugeordneten Typ, den Titel und die Beschreibung beschrieben wird. Abschließend werden die Ergebnisse in eine CSV-Datei exportiert, um eine einfache Weiterverarbeitung in nachfolgenden Schritten zu ermöglichen.

Diese Methode gewährleistet eine präzise und systematische Extraktion der relevanten Daten, indem sie sicherstellt, dass nur vollständig definierte Anforderungen mit Titel und Beschreibung in die Trainingsdaten einfließen. Durch den modularen Aufbau des Codes kann die Verarbeitung leicht an neue Anforderungen oder zusätzliche Datenquellen angepasst werden.

4.3.4 Modeling

Die Modeling Phase befasst sich mit der automatisierten Generierung von Test Cases auf Basis der in der Data Preparation Phase extrahierten Anforderungen. Hierbei wird die OpenAI GPT-4o-API genutzt, um die Beschreibungen der Anforderungen in strukturierte Test Cases zu überführen, die direkt in der Qualitätssicherung eingesetzt werden können. Diese strukturierte Generierung ermöglicht es, Anforderungen konsistent und präzise in Tests zu überführen, ohne dass eine manuelle Erstellung erforderlich ist. Python soll die Programmiersprache der Wahl sein. Python zeichnet sich im Vergleich zu anderen Programmiersprachen durch seine außergewöhnliche Flexibilität aus und eignet sich daher besonders für das Prototyping (Yuill und Halpin 2006). Außerdem bietet Python umfangreiche Standardbibliotheken, die eine schnelle Programmentwicklung ermöglichen. Die benötigten Python-Bibliotheken für die Implementierung sind in Tabelle 4-3 aufgelistet.

Zu Beginn der Modellierung werden die Anforderungen aus der vorbereiteten CSV-Datei in den Modellierungsprozess eingebunden. Jede Anforderung wird durch eine spezifische Eingabe erweitert, die das Modell anweist, Test Cases in einer festgelegten Struktur zu generieren. Diese Struktur umfasst unter anderem eine kurze Beschreibung des Test Cases, die benötigten Vorbedingungen, die auszuführenden Testschritte sowie deren erwartete Ergebnisse, Nachbedingungen und ergänzende Hinweise. Durch die Vorgabe einer einheitlichen Struktur wird sichergestellt, dass die generierten Test Cases eine einheitliche Struktur besitzen, die nachvollziehbar und leicht weiterzuverarbeiten sind. Der Prompt sollte folgendermaßen aufgebaut sein:

"Generate the test case including the test steps in the format **Test Case Title; **Test Description**; **Pre-Actions**; **Test Steps(Actions and Expected Results)** ; **Post-Actions**; **Remarks** for this requirement: " + req['title'] + req['description']**

Dieser Prompt stellt eine strukturierte und spezifische Anweisung dar, die darauf abzielt, ein LLM effektiv zur Generierung qualitativ hochwertiger Test Cases auf Basis von Anforderungen zu nutzen. Die Formulierung des Prompts zielt darauf ab, dass die Test Cases möglichst präzise und vollständig generiert werden. Dabei werden die einzelnen Bestandteile des Test Cases durch vordefinierte Abschnitte wie Test Case Title, Test Description, Pre-Actions, Test Steps (Actions and Expected Results), Post-Actions und Remarks eindeutig definiert. Diese Struktur entspricht der Struktur die für Anforderungen im ALM Tool Codebeamer hinterlegt sind, und gewährleistet, dass das LLM die erwarteten Inhalte systematisch und in einem einheitlichen Format bereitstellt. Dies erleichtert die weitere Verarbeitung und Auswertung der generierten Ergebnisse, insbesondere im Kontext von Evaluierungsmetriken, die nachfolgend erarbeitet werden.

Ein weiterer zentraler Aspekt des Prompts ist die Verwendung einer expliziten Beschreibung der Anforderungen durch die Variable req['description']. Durch diese Variable wird die Anforderungsbeschreibung, die zuvor extrahiert wird an den Prompt angehängt, wodurch die Test Cases automatisch generiert werden können. Diese Anforderungsbeschreibungen liefern dem LLM den notwendigen kontextuellen Input, um die Test Cases an die spezifischen Anforderungen anzupassen. Der Zusatz "for this requirement:" stellt sicher, dass die Ergebnisse im Kontext des Requirements Engineering verstanden werden. Die Einbindung von Test Steps (Actions and Expected Results) gibt dem LLM die Anweisung, nicht nur generische Testschritte zu erstellen, sondern auch die spezifischen Aktionen und erwarteten Ergebnisse explizit zu formulieren. Diese Detaillierung ist entscheidend, da sie den praktischen Nutzen der Test Cases erhöht, indem sie klare Anweisungen für die Durchführung und die erwarteten Ergebnisse bereitstellt. Dies trägt zu einer besseren Rückverfolgbarkeit zwischen den Anforderungen und den Test Cases bei. Die Verwendung eines definierten Formats mit markierten Abschnitten wie Test Case Title und Test Description schafft eine konsistente Struktur, die für die Nachvollziehbarkeit und Lesbarkeit der generierten Inhalte förderlich ist. Diese Strukturierung erleichtert zudem die automatisierte Nachbearbeitung der Ergebnisse, um ein Dokument zu erhalten, welches in das ALM Tool importiert werden kann.

Der Prompt adressiert die Anforderungen an die Generierung von Test Cases durch ein LLM, indem er klare Anweisungen, eine strukturierte Formatierung und eine inhaltliche Fokussierung auf die spezifischen Anforderungen kombiniert. Diese Eigenschaften fördern die Konsistenz, Relevanz und Praxistauglichkeit der generierten Test Cases und schaffen eine Grundlage für deren effiziente Nutzung und Evaluierung.

Das GPT-4o-Modell verarbeitet die Eingaben und generiert basierend auf den bereitgestellten Anforderungen vollständige Test Cases. Diese Ergebnisse werden anschließend einer Nachbearbeitung unterzogen, um die Importierbarkeit zurück ins ALM Tool sicherzustellen. Hierbei werden unerwünschte Zeichen entfernt, und die Inhalte werden auf eine klare und präzise Darstellung hin überprüft. Besonderes Augenmerk liegt dabei auf der Unterteilung der Testschritte in durchzuführende Aktionen und die zu erwartenden Ergebnisse, wodurch die praktische Anwendung der Test Cases erleichtert wird. Die abschließend bereinigten und strukturierten Test Cases werden in einem Excel-Format gespeichert. Diese tabellarische Darstellung bietet eine übersichtliche Organisation der Test Cases, die direkt mit den zugehörigen Anforderungen verknüpft sind. Das Excel-Format erlaubt zudem eine einfache Weiterverarbeitung in bestehenden ALM Tools und bietet eine klare Grundlage für die Zusammenarbeit zwischen verschiedenen Teams.

Tabelle 4-3 Anzuwendende Bibliotheken und ihre Erklärungen

Bibliothek	Erklärung
os	Die Bibliothek bietet Funktionen zur Interaktion mit dem Betriebssystem. Sie ermöglicht den Zugriff auf Betriebssystemressourcen wie Dateisysteme und Umgebungsvariablen sowie die Ausführung systembezogener Operationen.
pandas	Die Bibliothek beinhaltet Tools zur Analyse und Manipulation von Daten. Sie bietet flexible Datenstrukturen, die eine effiziente Verarbeitung und Analyse ermöglichen.
re	Eine Standardbibliothek für die Arbeit mit regulären Ausdrücken. Ermöglicht das Suchen, Prüfen und Ersetzen von Textmustern.
flask	Ein lightweight Web-Framework für Python, das die Erstellung von Webanwendungen erleichtert.
openai	Eine offizielle Bibliothek zur Interaktion mit der OpenAI-API. Wird verwendet, um GPT-Modelle für NLP-Aufgaben zu nutzen.
dotenv	Eine Bibliothek, die das Laden von Umgebungsvariablen aus einer .env-Datei ermöglicht, um sensible Daten wie API-Schlüssel zu schützen.

Die Bibliothek os wird verwendet, um plattformunabhängig mit Dateipfaden und Verzeichnissen zu arbeiten. Insbesondere wird sichergestellt, dass benötigte Verzeichnisse für Uploads und Ausgaben existieren, und es wird eine sichere und konsistente Verwaltung von Dateipfaden ermöglicht. Pandas dient der Verarbeitung und Manipulation tabellarischer Daten, wie dem Lesen, Bearbeiten und Speichern von CSV- und Excel-Dateien.Dies ist erforderlich für die Verwaltung der Anforderungen und generierten Test Cases. Die Bibliothek re (Regular Expressions) wird eingesetzt, um Textmuster in den generierten Test Cases zu identifizieren, Abschnitte zu extrahieren oder Inhalte gezielt zu bereinigen, um die Struktur und Lesbarkeit der Daten zu gewährleisten. Die Webanwendung wird durch die Bibliothek flask realisiert, die Funktionen für die Verarbeitung von HTTP-Anfragen, die Bereitstellung von Webschnittstellen sowie das Hoch- und Herunterladen von Dateien bereitstellt. Dies ermöglicht eine interaktive Nutzung der Anwendung durch die Nutzer. Die Bibliothek openai wird genutzt, um die Generierung von Test Cases mittels einer Schnittstelle zur OpenAI-API durchzuführen. Dabei wird die API für die Verarbeitung der Anforderungen in natürliche Sprache und die Erstellung strukturierter Test Cases eingebunden. Schließlich wird die Bibliothek dotenv verwendet, um Umgebungsvariablen aus einer Konfigurationsdatei (key.env) sicher zu laden, insbesondere für die Authentifizierung mit der OpenAI-API. Dies stellt sicher, dass sensible Informationen wie API-Schlüssel nicht direkt im Quellcode enthalten sind, und somit nicht jeder, der Zugriff auf den Code hat auch Zugriff auf sensible Daten hat.

Durch den kombinierten Einsatz dieser Bibliotheken wird eine flexible, funktionale und plattformunabhängige Anwendung ermöglicht, die sowohl datenbezogene Operationen als auch die Interaktion mit Nutzern und externen Schnittstellen unterstützt. Die entwickelte Lösung ermöglicht die effiziente Generierung hochwertiger Test Cases aus Anforderungen und schafft eine skalierbare Grundlage für den Einsatz in realen Anwendungsfällen. Die Automatisierung dieses Prozesses stellt eine erhebliche Zeitersparnis dar und minimiert gleichzeitig potenzielle Fehler, die bei der manuellen Test Case Erstellung auftreten könnten.

4.3.5 Evaluation und Deployment

Nach der Modellierung folgt die Evaluation Phase, in der die generierten Test Cases auf ihre Qualität hin überprüft werden. Die Modelle werden genutzt, um Test Cases basierend auf den Anforderungen zu generieren. Diese generierten Test Cases werden anschließend mit den ursprünglichen Test Cases und den Ergebnissen eines spezialisierten Tools verglichen, um ihre Genauigkeit und Relevanz zu bewerten. Diese Evaluierung soll ermitteln, ob die generierten Test Cases den Anforderungen entsprechen und in der Praxis anwendbar sind. Die Methodik für die Evaluierung wird in Kapitel 4.4 tiefergehend erläutert.

In der letzten Phase, dem Deployment, wird das entwickelte Modell in die Praxis umgesetzt. Ein wichtiger Aspekt dieser Phase ist die Entwicklung einer benutzerfreundlichen grafischen Oberfläche (GUI). Diese GUI ermöglicht es den Anwendern, ReqIF-Dateien hochzuladen und automatisch generierte Test Cases im ReqIF-Format zu erhalten. Dabei wird auch die Verlinkung zwischen Anforderungen und Test Cases berücksichtigt, um die Nachverfolgbarkeit zu gewährleisten und eine konsistente Dokumentation sicherzustellen.

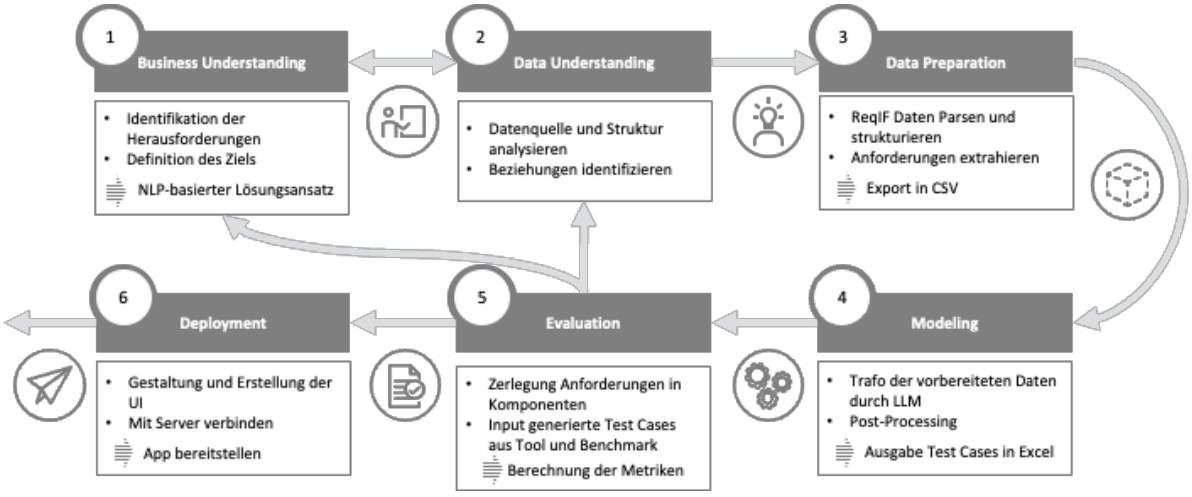


Abbildung 4-6 Abgewandeltes Vorgehensmodell

Zusammenfassend ist der Prozess nochmal in Abbildung 4-6 dargestellt. Die Integration von NLP in die Test Case Automatisierung durch das CRISP-DM-Modell stellt eine systematische und effektive Herangehensweise dar. Durch den Einsatz von Low-Shot Learning können NLP-Modelle flexibel eingesetzt werden, selbst wenn nur begrenzte Trainingsdaten zur Verfügung stehen. Insbesondere das Low-Shot Learning bietet eine praktikable Lösung, um mit wenigen Anforderungen und Test Cases qualitativ hochwertige und relevante Test Cases zu generieren.

4.4 Evaluierungskonzept

Die Evaluierung und Validierung der Ergebnisse des Demonstrators soll in zwei Phasen stattfinden. Dafür wird in diesem Kapitel ein Konzept entwickelt, anhand dessen die Implementierung bewertet werden kann. Die Methodik der Implementierung wurde in Kapitel 4.3 erläutert und die Umsetzung wird in Kapitel 5.1.2 beschrieben. Für die erste Phase wird die Qualität der Ergebnisse nach den Kriterien bewertet, die in Tabelle 4-4 dargestellt sind.

Tabelle 4-4 Formeln der Bewertungsmetriken (In Anlehnung an Seidel und Späthe (2024, S.143))

Kennzahl	Formel
Precision	$\frac{Echt\ Positiv\ (EP)}{EP + Falsch\ Positiv\ (FP)}$
Recall	$\frac{EP}{EP + Falsch\ Negativ\ (FN)}$
F1-Score	$2 \times \frac{Precision \times Recall}{Precision + Recall}$
Strukturelle Konsistenz	$SC(T_1, T_2) = 1 - (\frac{\lvert S(T_1) - S(T_2)\rvert}{\max(S(T_1), S(T_2))} + (1 - Schrittübereinstimmung)$
Konsistenz	$Ähnlichkeit(R_1, R_2) - SC(T_1, T_2)$

Die Kennzahlen Precision, Recall und der F1-Score werden häufig für die Evaluierung von Machine Learning Anwendungen verwendet, wovon NLP ein Unterthema ist (Goutte und Gaussier 2005; Seidel und Späthe 2024). Um die Evaluation durchzuführen, werden die Anforderungen in ihre wesentlichen Bestandteile zerlegt. Diese beinhalten in der Regel Akteur (z.B. der Nutzer oder das System, das eine Aktion ausführt), Aktion (die spezifische Handlung, die durchgeführt werden soll), Bedingung/Akzeptanzkriterium (Einschränkungen oder Rahmenbedingungen, unter denen die Aktion stattfinden soll), Ziel (das gewünschte Ergebnis der Anforderung) sowie technische Details (wie eine vorgegebene Leistung, Geschwindigkeit oder Ähnliches), Abhängigkeiten (zusammenhängende Anforderungen oder Test Cases) und ein Edge Case (seltener aber möglicher Sonderfall, der zum Versagen führen würde). Diese Zerlegung ermöglicht es, alle Aspekte der Anforderung, die im Test Case überprüft werden sollten abzudecken, um die Precision, Recall und F1-Score zu berechnen.

Precision bewertet die Genauigkeit der generierten Test Cases, indem geprüft wird, wie viele der im Test Case enthaltenen Schritte tatsächlich relevant und korrekt in Bezug auf die Anforderung sind. Wie in Tabelle 4-4 zu sehen ist, berechnet sich die Precision als das Verhältnis der Echt Positiven (EP) zu allen vorkommenden Komponenten, der Summe aus EP und Falsch Positiven (FP). EP sind hier Komponenten, die sowohl in der Anforderung und auch im Test Case vorkommen. FP sind Komponenten, die im Test Case vorkommen, aber weder explizit noch implizit in der Anforderung vorkommen. Precision kann Werte zwischen 0 und 1 annehmen, wobei:

- Ein Wert von 1 bedeutet, dass alle Schritte im Test Case relevant und korrekt sind (perfekte Genauigkeit).
- Ein Wert von 0 zeigt an, dass keine der enthaltenen Komponenten relevant ist.

Ein hoher Precision-Wert ist ein Indikator dafür, dass das Tool nur die notwendigen und relevanten Testschritte erzeugt, ohne überflüssige Schritte hinzuzufügen.

Recall misst die Vollständigkeit der generierten Test Cases, indem überprüft wird, ob alle relevanten Aspekte der Anforderung im Test Case abgedeckt sind. Wie in Tabelle 4-4 dargestellt, wird Recall berechnet, indem die Anzahl der relevanten Komponenten, die im Test Case korrekt berücksichtigt sind (Echt Positive), zur Gesamtanzahl der in der Anforderung spezifizierten relevanten Komponenten ins Verhältnis gesetzt wird. Recall kann ebenfalls Werte zwischen 0 und 1 annehmen:

- Ein Wert von 1 bedeutet, dass alle wichtigen Aspekte der Anforderung im Test Case enthalten sind (perfekte Vollständigkeit).

- Ein Wert von 0 zeigt, dass kein relevanter Aspekt der Anforderung abgedeckt wurde.

Ein hoher Recall-Wert zeigt die Fähigkeit des Tools, umfassende Test Cases zu erstellen, die alle wesentlichen Bestandteile der Anforderungen berücksichtigen.

Der F1-Score ist eine kombinierte Kennzahl, die Precision und Recall zu einer einzigen Metrik zusammenfasst. Wie in Tabelle 4-4 dargestellt ist, wird der F1-Score als harmonisches Mittel von Precision und Recall berechnet. Diese Metrik misst das Gleichgewicht zwischen Präzision und Vollständigkeit. Der F1-Score kann ebenfalls Werte zwischen 0 und 1 annehmen:

- Ein Wert von 1 bedeutet, dass sowohl Precision als auch Recall perfekt sind (perfekte Balance).
- Ein Wert von 0 zeigt an, dass entweder Precision oder Recall (oder beide) sehr niedrig sind, und deutet auf eine schlechte Leistung des Tools hin.

Ein hoher F1-Score zeigt die Fähigkeit des Tools, Test Cases zu generieren, die sowohl präzise als auch vollständig sind. Ein niedriger Wert deutet auf ein Ungleichgewicht hin, bei dem entweder wichtige Aspekte fehlen oder unnötige Schritte enthalten sind.

Des Weiteren wird die Konsistenz-Kennzahl eingeführt, um die Qualität der Test Cases zu beurteilen. Die Berechnung dieser Kennzahl erfolgt auf Basis der strukturellen Konsistenz und der semantischen Ähnlichkeit zwischen den Anforderungen und den dazugehörigen Test Cases. Ziel ist es, die Übereinstimmung zwischen verschiedenen Test Cases und deren zugrunde liegenden Anforderungen zu analysieren. Die strukturelle Konsistenz misst, wie ähnlich die Struktur der Test Cases für ähnliche Anforderungen ist. Sie wird durch den Vergleich der Anzahl der Test Steps zweier Test Cases bestimmt. Wie in Tabelle 4-4 zu sehen ist, wird die relative Differenz der Anzahl an Test Steps berücksichtigt, um eine Metrik zu erzeugen, die unabhängig von der absoluten Anzahl der Schritte ist. Abweichungen führen zu einer proportionalen Reduktion der Konsistenz, wobei größere Abweichungen einen stärkeren Einfluss auf die Bewertung haben. Diese Metrik kann ebenfalls Werte zwischen 0 und 1 annehmen:

- Ein Wert von 1 bedeutet, dass die Test Cases für ähnliche Anforderungen eine identische Struktur aufweisen.
- Ein Wert von 0 zeigt an, dass es keine Übereinstimmung in der Struktur der Test Cases gibt.

Für die Auswertung dient die strukturelle Konsistenz als Hilfsvariable zur Berechnung der Konsistenz der Test Cases. Die Konsistenz-Kennzahl bewertet die allgemeine Ähnlichkeit der generierten Test Cases für ähnliche Anforderungen. Wie in Tabelle 4-4 dargestellt, basiert die Berechnung auf dem Durchschnitt der Ähnlichkeitswerte aller Test Case Paare. Untersucht wird, ob die Konsistenz von Test Cases bei Anforderungen mit einer Ähnlichkeit von mindestens 0,5 überwiegend positiv ist. Damit wird die Grundlage für weitergehende Untersuchungen geschaffen, ob die negativen Konsistenzwerte auf spezifische Muster oder Ursachen zurückzuführen sind. Die Ähnlichkeit wird über die Cosine Similarity überprüft, welche wie folgend berechnet wird:

$$Cosine\ similarity = \cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\| \boldsymbol{A} \| \cdot \| \boldsymbol{B} \|}$$

Wobei A und B vektorisierte Wörter oder Textabschnitte darstellen, von denen im Zähler das Skalarprodukt berechnet wird. Im Nenner werden die Normen der Vektoren multipliziert, welche die Längen der Vektoren repräsentieren. Diese Normen skalieren den Wert des Skalarprodukts, sodass die Cosine Similarity unabhängig von der Länge der Vektoren bleibt. Eine Cosine Similarity nahe 1 bedeutet, dass die Wörter oder Textabschnitte sich sehr ähnlich sind, wogegen eine Cosine Similarity nahe 0 auf eine geringe Ähnlichkeit hinweist.

Insgesamt kann die Konsistenz-Kennzahl Werte zwischen -1 und 1 annehmen:

- Positiv konsistente Test Cases (≥ 0) zeigen harmonisierte Strukturen und Beschreibungen.

- Negativ konsistente Test Cases (< 0) weisen größere strukturelle Abweichungen oder Unstimmigkeiten in der Formulierung auf.
- Kritisch inkonsistente Test Cases (Konsistenz ≤ -0,5) können spezifische Schwachstellen in der Test Case-Entwicklung aufzeigen, die gezielte Verbesserungsmaßnahmen erfordern.

Eine hohe Konsistenz zeigt die Fähigkeit des Tools, ähnliche Anforderungen einheitlich zu behandeln und ist wichtig für die Qualitätssicherung und die Vermeidung von Redundanzen.

Um diese Kennzahl zu ermitteln, werden folgende Bibliotheken für Python verwendet:

Tabelle 4-5 Verwendete Bibliotheken zur Evaluierung

Bibliothek	Erklärung
Scikit-learn	Die Bibliothek beinhaltet Werkzeuge für maschinelles Lernen. Sie stellt Algorithmen und Funktionen für Datenvorverarbeitung, Modelltraining, Evaluierung und Vorhersagen bereit.
pandas	Siehe Tabelle 4-3

Die Cosine Similarity lässt sich mit Hilfe der TF-IDF-Vektorisierung, beides aus Scikit-learn, berechnen. Hierbei werden sowohl die Anforderungen als auch die Test Cases in numerische Vektoren überführt, die den Inhalt und die Häufigkeit der Begriffe innerhalb des jeweiligen Textes repräsentieren. Die Cosine Similarity misst dabei den Winkel zwischen den Vektoren, wobei ein höherer Wert auf eine stärkere semantische Ähnlichkeit hinweist. Die Konsistenzkennzahl kombiniert die strukturelle Konsistenz und die semantische Ähnlichkeit, indem sie die Differenz zwischen beiden Werten berechnet. Diese Differenz soll aufzeigen, wie stark strukturelle Diskrepanzen durch inhaltliche Ähnlichkeiten ausgeglichen werden können oder ob die Test Cases sowohl inhaltlich als auch strukturell konsistent sind. Die Berechnungen werden für jedes Paar an Anforderungen und Test Cases durchgeführt, deren Ähnlichkeit einen definierten Schwellenwert überschreitet.

Zusammenfassend wird die Qualität der generierten Test Cases anhand einer Kombination verschiedener Kennzahlen bewertet, die sowohl die semantische als auch die strukturelle Dimension der Test Cases berücksichtigen. Precision, Recall und F1-Score dienen dazu, die Genauigkeit, Vollständigkeit und Balance der Test Cases zu quantifizieren, indem sie die Übereinstimmung zwischen den Anforderungen und den Test Cases analysieren. Ergänzend dazu bewerten die strukturelle Konsistenz und die allgemeine Konsistenz, wie einheitlich und harmonisiert ähnliche Anforderungen im Tool verarbeitet werden. Diese Kennzahlen ermöglichen eine umfassende Analyse, um die semantische Präzision, die strukturelle Einheitlichkeit und die allgemeine Konsistenz der generierten Test Cases zu beurteilen, und liefern eine fundierte Grundlage für die Identifikation von Verbesserungspotenzialen.

5 Anwendung und Evaluierung der Integration

In diesem Kapitel wird die praktische Anwendung und Evaluierung der Integration von Natural Language Processing in Application Lifecycle Management beschrieben. Dabei liegt der Fokus auf den Ergebnissen der Implementierung des entwickelten Tools zur Generierung von Test Cases sowie auf dessen Leistungsbewertung im Vergleich zu einem spezialisierten Tool. Ziel ist es, die Effektivität und Effizienz der NLP-Integration zu analysieren und deren Potenzial für den praktischen Einsatz im Anforderungsmanagement und der Qualitätssicherung aufzuzeigen. Zusätzlich werden Herausforderungen und Grenzen der aktuellen Ansätze aufgezeigt, um Optimierungsmöglichkeiten für zukünftige Entwicklungen abzuleiten.

5.1 Exemplarische Anwendung des Use Cases

Für die exemplarische Anwendung des Use Cases wird die Methodik, die in Kapitel 4.3 entwickelt wird, angewandt. Hierfür wird in diesem Kapitel zunächst auf den Beispieldatensatz eingegangen, der für die Umsetzung verwendet wird. Danach wird die Umsetzung der beschriebenen Schritte dargestellt und beschrieben, um im darauffolgenden Kapitel die Ergebnisse zu evaluieren.

5.1.1 Beschreibung des Datensatzes

Der Datensatz für das Testen und die Evaluierung des Tools wird aus einem Codebeamer-Projekt exportiert und umfasst insgesamt 28 Anforderungen in englischer Sprache. Diese Anforderungen sind in drei Kategorien gegliedert: Customer Requirement Specifications, System Requirements und Software Requirements. Die Einteilung in diese Kategorien spiegelt unterschiedliche Perspektiven und Ebenen der Anforderungsdefinition wider, die von den kundenseitigen Erwartungen bis hin zu spezifischen technischen Details reichen.

Jede Anforderung im Datensatz ist durch eine eindeutige ID gekennzeichnet. Diese Identifier ermöglichen eine klare Zuordnung und Nachverfolgbarkeit der Anforderungen, insbesondere in Prozessen, die eine strukturierte Verwaltung und Verlinkung von Informationen erfordern. Neben der ID umfasst der Datensatz für jede Anforderung einen Titel, der die wesentlichen Inhalte prägnant zusammenfasst. Der Titel dient als erster Orientierungspunkt, während die dazugehörige Beschreibung detaillierte Informationen zu Inhalt, Ziel und Kontext der jeweiligen Anforderung bereitstellt. Die Beschreibungen sind ausführlich und standardisiert formuliert, wobei häufig die Perspektive eines zentralen Stakeholders, beispielsweise des Original Equipment Manufacturers (OEM), eingenommen wird. In diesen Beschreibungen wird konkretisiert, welche Funktionen, Eigenschaften oder Ergebnisse von den jeweiligen Produkten erwartet werden. Dies kann sowohl technische als auch organisatorische Aspekte umfassen, wie etwa funktionale Spezifikationen, Leistungsparameter oder Rahmenbedingungen für die Umsetzung. Die Anforderungen decken verschiedene thematische Bereiche ab und beschreiben unterschiedlich komplexe Sachverhalte. Innerhalb des Datensatzes finden sich Anforderungen, die beispielsweise funktionale Eigenschaften von Systemen, Dokumentationsvorgaben oder spezifische Parameter technischer Implementierungen betreffen. Die Gliederung in die drei Hauptkategorien bietet dabei eine klare Struktur, die eine thematische Zuordnung der Inhalte ermöglicht.

5.1.2 Umsetzung des Use Cases

Nach der in Kapitel 4.3 beschriebenen Methodik wird der Use Case in Visual Studio Code mit Python, HTML, Javascript und CSS implementiert. Als Vorbild für die Optik wurde die Unternehmenswebsite des Kooperationspartners gewählt, die mit Hilfe der Toolbox von Flask umgesetzt wurde. Das Ziel des Designs der Webanwendung ist eine klar strukturierte Benutzeroberfläche, die sowohl funktional auch visuell ansprechend gestaltet ist. Die Oberfläche ist in Abbildung 5-1 zu sehen.

Im zentralen Bereich der Seite befindet sich die Upload-Komponente, die durch eine div-Struktur mit der Klasse „Upload-Container“ realisiert wird. Diese Komponente ist so gestaltet, dass sie zentriert auf der Seite positioniert ist. Das CSS sorgt mit einem weißen Hintergrund, einem grünen Rahmen von 3 Pixeln Stärke und abgerundeten Ecken für eine klare Abgrenzung vom restlichen Seiteninhalt. Zusätzlich erzeugt ein leichter Schattierungseffekt eine visuelle Tiefe. Der Container enthält einen Ordner Icon, das im oberen Bereich angezeigt wird und mithilfe von CSS auf eine Breite von 100 Pixeln skaliert ist. Dieses Bild dient als visuelle Unterstützung für die Dateiauswahl. Darunter wird der Benutzer durch zwei Absätze

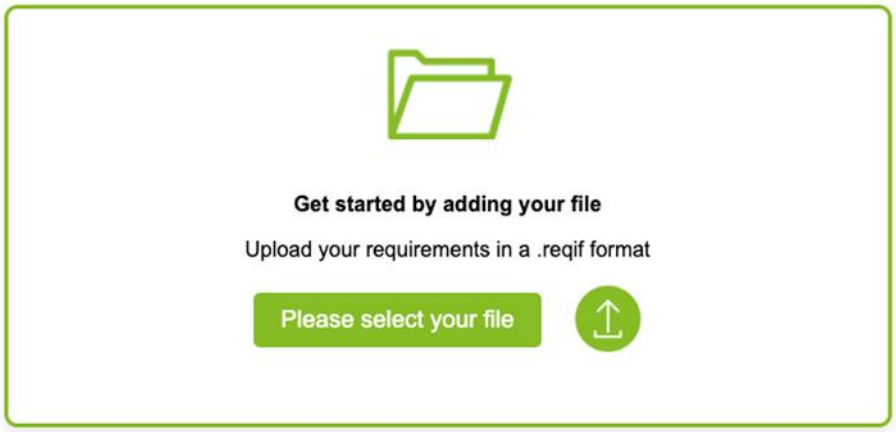


Abbildung 5-1 Graphical User Interface der exemplarischen Umsetzung

mit erklärendem Text in den Upload-Prozess eingeführt. Diese Textelemente werden mittig ausgerichtet, um eine einheitliche Struktur im Container zu gewährleisten. Das Herzstück der Upload-Komponente ist das Upload-Formular, das durch ein Input-Feld für die Dateiauswahl und eine klare Beschriftung in Form eines Labels realisiert wird. Das CSS sorgt dafür, dass das Label durch einen grünen Hintergrund, weiße Schriftfarbe und abgerundete Ecken hervorsticht, und seine Interaktivität betont. Ergänzt wird dies durch eine Upload-Schaltfläche in Form eines Icons, die den Upload-Vorgang abschließt. Dieses Icon ist ein Bild, das mit CSS transparent gestaltet wird, sodass es sich nahtlos in das Layout einfügt. Wenn der „Please select your file“ Button geklickt wird, öffnet sich ein Dateiauswahl-Dialogfenster in dem die lokal abgelegte ReqIF-Datei ausgewählt werden kann. Sobald eine Datei ausgewählt ist, erscheint diese in der GUI, wird aber nicht direkt hochgeladen, damit überprüft werden kann, ob die richtige Datei ausgewählt wurde. Dieser Schritt ist in Abbildung 5-2 abgebildet. Der tatsächliche Upload wird dann durch Klicken des Upload Icons initialisiert.

Im Backend wird die hochgeladene Datei abgefragt. Zunächst wird überprüft, ob eine Datei hochgeladen wurde und ob sie einen gültigen Namen besitzt. Falls diese Voraussetzungen erfüllt sind, wird die Datei in einem temporären Verzeichnis zwischengespeichert. Dieses Verzeichnis, definiert durch die Konfiguration Upload_Folder, wird vorab geprüft und gegebenenfalls mithilfe der Methode os.makedirs erstellt, um sicherzustellen, dass der Speicherort verfügbar ist. Der vollständige Pfad der hochgeladenen Datei wird dynamisch mit os.path.join erzeugt. Anschließend wird die Datei an diesem Speicherort abgelegt. Zusätzlich wird der Pfad zur hochgeladenen Datei in der Flask-Session gespeichert. Die Session fungiert dabei als temporärer Speicher, der es ermöglicht, benutzerspezifische Informationen während der Interaktion mit der Anwendung bereitzuhalten. Durch diese Vorgehensweise bleibt der Pfad zur Datei für nachfolgende Verarbeitungsschritte, wie die

Generierung von Test Cases, verfügbar, ohne dass der Benutzer erneut Eingaben vornehmen muss.

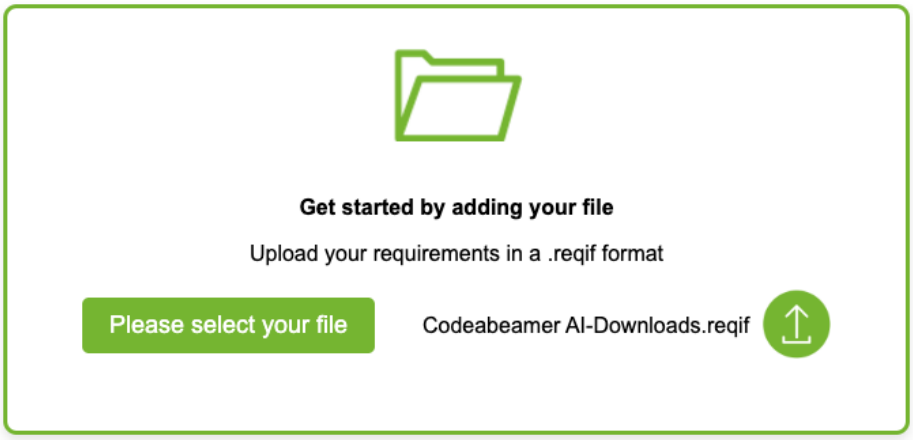


Abbildung 5-2 Graphical User Interface nach Dateiauswahl

Nach Abschluss des Uploads wird ein dynamisch generierter Button für die Generierung der Test Cases eingeblendet. Dieser Button wird durch eine CSS-Klasse mit einem einheitlichen grünen Design gestaltet, das auf die gesamte Webanwendung abgestimmt ist, wie in Abbildung 5-3 zu sehen ist.

Während der Generierung wird der Button durch eine Ladeanimation ersetzt, die durch das Processing-Element realisiert wird. Dieses Element verwendet eine Kombination aus einem stilisierten Button und einer animierten GIF-Datei, die mithilfe von CSS zentriert und in einer flexiblen Box ausgerichtet werden. Der Prozess der Test Case Generierung dauert einige Minuten. Damit der Benutzer weiß, dass noch Aktionen stattfinden, wurde die Animierung gewählt. Bei zu langer Inaktion auf der GUI könnte der Benutzer auf die Idee kommen, dass die Seite eingefroren ist, und das Fenster vor Beendigung des Vorgangs schließen. Dieser Zustand wird in Abbildung 5-4 dargestellt.

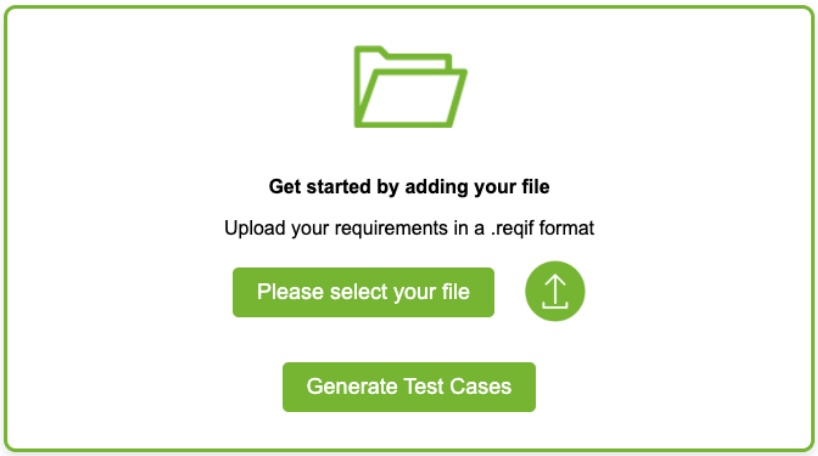


Abbildung 5-3 Graphical User Interface nach Upload der Datei

Der Prozess der Test Case-Generierung in der Webanwendung erfolgt in drei Hauptschritten: der Verarbeitung der Anforderungen, der Generierung der Test Cases und der Formatierung der Ergebnisse für die Ausgabe. Wie in Kapitel 4.3.3 beschrieben, werden zunächst die Anforderungen aus der hochgeladenen .reqif-Datei extrahiert. Dabei werden die Inhalte der Datei mithilfe einer XML-Verarbeitung analysiert. Die relevanten Informationen werden aus den XML-Elementen und Attributen identifiziert und in die Bestandteile id, type, title und description unterteilt. Diese Daten werden anschließend in einer CSV-Datei gespeichert, die als Zwischenformat für die weiteren Verarbeitungsschritte

dient. Eine Zeile dieser Datei sieht je nach Programm, mit dem es geöffnet wird, folgendermaßen aus:

ID	Type	Title	Description
CB-76e022bf-c871-482a-a657-839cc52e5196	Software Requirements	Object Tracking	"The system SW shall track objects based on their detected colors by using a PID controller, in case the deviation of the image center is >= 15 pixels"

Das Format CSV ist ein Dateiformat, welches die Werte durch Kommata voneinander trennt, wodurch nicht jedes Programm die Ergebnisse als Tabelle mit mehreren Spalten wiedergeben wird. Dieses Format kann aber sehr gut an den nächsten Programmschritt weitergegeben werden und enthält alle Informationen, die für die Generation von Test Cases benötigt werden.

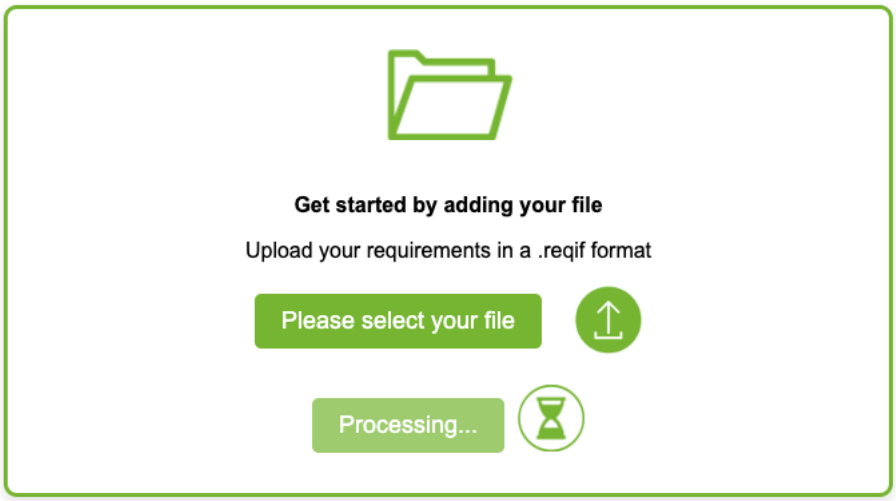


Abbildung 5-4 Graphical User Interface nach Start des Test Case Generierungsvorgangs

Im nächsten Schritt erfolgt dann die Generierung der Test Cases auf Grundlage der zuvor extrahierten Anforderungen. Die extrahierten Daten werden aus der CSV-Datei geladen und für jede Anforderung wird eine Eingabe für das Large Language Model erstellt. Diese Eingabe besteht aus der Beschreibung der jeweiligen Anforderung und einem festgelegten Prompt, der die Struktur des gewünschten Test Cases vorgibt. Das LLM erzeugt daraufhin vollständige Test Cases, die für jede Anforderung spezifisch sind. Die generierten Test Cases werden in einer neuen CSV-Datei gespeichert. Wie diese strukturiert ist, beziehungsweise wie eine Ausgabe aussieht, ist im Anhang unter Tabelle A-7-2 zu finden. Im letzten Schritt werden die generierten Test Cases für die Ausgabe in einem Excel-Format aufbereitet. Hierbei werden die Inhalte der Test Cases analysiert und in ihre verschiedenen Abschnitte zerlegt: der Titel, die Pre-Actions, die Test Steps, die Post-Actions und die Remarks. Für die Test Steps erfolgt eine zusätzliche Unterteilung in Aktionen und erwartete Ergebnisse, wobei die einzelnen Schritte nummeriert und detailliert aufbereitet werden. Die strukturierten Daten werden in eine tabellarische Form gebracht, die zusätzliche Informationen wie die Verknüpfung zur ursprünglichen Anforderung enthält. Abschließend werden die aufbereiteten Daten in einer Excel-Datei gespeichert, die dem Benutzer als klar strukturierte und benutzerfreundliche Ausgabe zur Verfügung gestellt wird.

Zu sehen, am unteren Ende der Upload-Komponente befindet sich der Download-Bereich, der erst nach erfolgreicher Generierung der Test Cases angezeigt wird. Der Download-Button wird im selben Stil wie der Generierungs-Button dargestellt, wodurch eine visuelle Konsistenz entsteht. Dargestellt ist dieser Zustand in Abbildung 5-5. Die gesamte Seite wird durch ein globales CSS-Design unterstützt, das die Grundstruktur des Dokuments mit einem weißen Hintergrund und der Schriftart "Arial, sans-serif" definiert. Diese Grundlage sorgt für eine klare und moderne Gestaltung, die durch gezielte Farbakzente in Grün und Schwarz ergänzt wird. Die Kombination aus HTML-Elementen und CSS-Stilen schafft so eine Webanwendung, die sowohl funktional als auch visuell ansprechend ist und den Benutzer effizient durch den Prozess der Test Case Generierung führt.

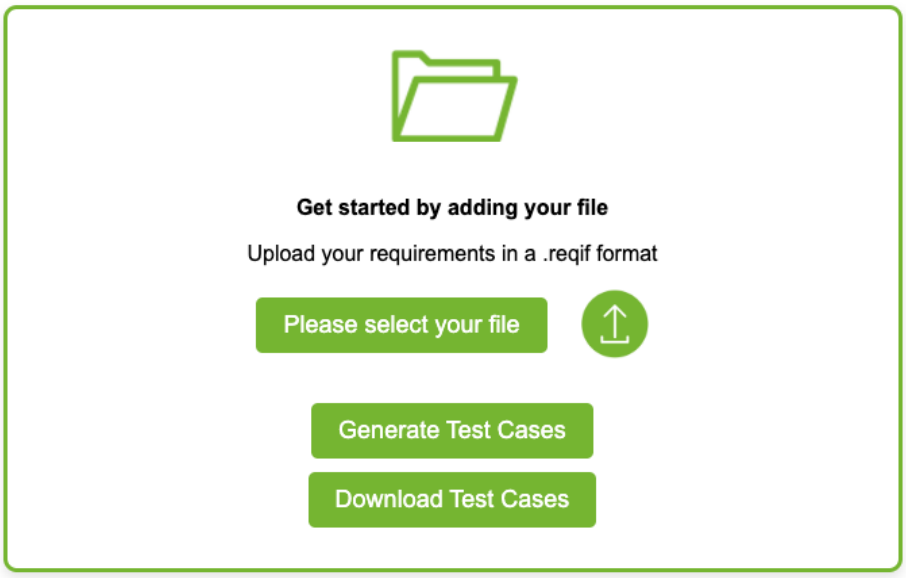


Abbildung 5-5 Graphical User Interface nach Beendigung des Test Case Generierungsvorgangs

Der Output der Anwendung ist eine Excel-Datei, die die generierten Test Cases in einer klar strukturierten Form darstellt, wie bereits in Kapitel 4.3 erläutert. Die Datei enthält Spalten, die wesentliche Bestandteile der Test Cases abbilden, darunter die zugehörige Anforderung, den Titel des Test Cases, eine Beschreibung, Maßnahmen vor der Durchführung (Pre-Actions), die Testschritte (aufgeteilt in Aktionen und erwartete Ergebnisse), Maßnahmen nach der Durchführung (Post-Actions) und zusätzliche Anmerkungen (Remarks). Durch die Spalte mit den Titeln der zugehörigen Anforderung können die Test Cases beim Import in Codebeamer direkt mit den Anforderungen verlinkt werden. Nach der Generierung wird die Excel-Datei in einem temporären Verzeichnis der Anwendung gespeichert, das speziell für die Zwischenspeicherung von Ergebnissen vorgesehen ist. Dieses Verzeichnis wird, wie für die Upload-Dateien, bei Bedarf erstellt, um sicherzustellen, dass ein Speicherort verfügbar ist. Der Pfad zur generierten Datei wird gespeichert, sodass er für den Benutzer spezifisch abrufbar ist. Der Download der Datei wird über einen dedizierten Endpunkt ermöglicht, der durch eine GET-Anfrage aufgerufen wird. Beim Abruf wird geprüft, ob die Datei verfügbar ist. Falls dies der Fall ist, wird sie dem Benutzer direkt zur Verfügung gestellt. Die Datei wird dabei so übertragen, dass sie zur lokalen Speicherung als Download angeboten wird. Falls keine Datei verfügbar ist, wird eine Fehlermeldung zurückgegeben. Der technische Ablauf umfasst die Erstellung der Excel-Datei während der Test Case-Generierung, die Speicherung in einem temporären Verzeichnis und die Bereitstellung über einen Endpunkt der Anwendung. Diese klare Trennung zwischen Generierung, Zwischenspeicherung und Bereitstellung gewährleistet eine effiziente und sichere Bereitstellung des Outputs, während gleichzeitig eine benutzerfreundliche Handhabung ermöglicht wird.

In Tabelle 5-1 ist ein Beispiel für die Ausgabe der Test Case Generierung für eine Anforderung zu finden. Das Format ist für diese Darstellung transformiert, um die Lesbarkeit an dieser Stelle

zu verbessern. Die Bezeichner sind im tatsächlichen Dokument in einer Zeile, statt in einer Spalte.

Tabelle 5-1 Beispielaufbau eines generierten Test Cases

Verifies	Collision Avoidance (break)
Title	Test Collision Avoidance (break)
Description	Collision Prevention Mechanism Verify that the system stops the vehicle before a collision occurs when dodging is not feasible.
Pre-Actions	1. Ensure the vehicle is in operational mode. 2. Verify that the vehicle's sensors (LIDAR, ultrasonic, cameras, etc.) are functioning and calibrated. 3. Set up a controlled environment with obstacles positioned in the vehicle's path to simulate potential collision scenarios. 4. Ensure that the vehicle's software is updated to the latest version implementing the collision prevention algorithms.
Test Steps Actions	1. Action Start the vehicle and drive towards a stationary obstacle at a moderate speed (e.g., 20 km/h). 2. Action Control the vehicle's path to ensure that there is no feasible way to dodge the obstacle (e.g., blocking the adjacent lanes). 3. Action Observe the vehicle's behavior as it approaches the obstacle. 4. Action Continue to monitor the distance between the vehicle and the obstacle. 5. Action Reduce the speed of the vehicle and repeat the approach with different scenarios, including varying obstacle sizes and distances. 6. Action Test with a moving obstacle introduced suddenly into the vehicle's path.
Test Steps Expected Results	1. The vehicle is in motion towards the obstacle. 2. The system detects that dodging options are unavailable. 3. The vehicle should begin to apply brakes automatically. 4. The vehicle stops before reaching the obstacle, avoiding any collision. 5. The outcome remains consistent; the vehicle stops before a potential collision regardless of the obstacle's size and the speed. 6. The vehicle identifies the impending collision and comes to a complete stop before impact if dodging options are not available.
Post-Actions	1. Document the results of each test scenario. 2. Perform a safety inspection of the vehicle's braking system and collision prevention sensors post-testing. 3. Reset the vehicle's systems and clear any test settings to return to normal operational mode.
Remarks	Ensure iterations are run multiple times with variations in speed and obstacle positioning to thoroughly validate the system's reliability in

	preventing collisions. Note any anomalies and conduct a review of the vehicle's software logs for additional diagnostics.

Die Ergebnisse legen nahe, dass der Algorithmus in der Lage ist, Test Cases zu generieren, die nicht nur die Anforderungen adressieren, sondern auch praxisrelevante Testbedingungen berücksichtigen. Die im Prompt definierte Struktur trägt dazu bei, die Bewertung der Testergebnisse zu standardisieren. Dennoch könnten Aspekte wie die Integration spezifischer Grenzbedingungen oder Randfälle noch stärker betont werden, um sicherzustellen, dass der generierte Test Case alle denkbaren Szenarien abdeckt. Insgesamt spiegelt der Aufbau ein hohes Maß an Konsistenz und Anwendbarkeit wider und unterstreicht damit das Potenzial des Tools für die Automatisierung von Testfällen.

5.2 Vergleichende Performanceanalyse und Validierung

Für die Durchführung der vergleichenden Performanceanalyse and Validierung wird die in Kapitel 4.4 erarbeitete Methodik angewandt. Als ersten Schritt müssen die in Kapitel 5.1 extrahierten Anforderungen in ihre Komponenten aufgeteilt werden. Dafür wird aus jeder vorhandenen Kategorie eine Anforderung kurz als Beispiel vorgestellt.

Customer Requirement Specification**: „ As the OEM, I want to have a documentation for the OTA updates history.“**	
Akteur:	„OEM“
Aktion:	„want to have a documentation“
Bedingung/ Akzeptanzkriterium:	„ for the OTA updates history “
Ziel/Erwartetes Ergebnis:	Transparency over update history
Technische Details:	--
Abhängigkeiten:	OTA updates
Edge Case:	No Documentation

Hier ist der OEM der verantwortliche Akteur, der sich um die Umsetzung dieser Anforderung kümmert. Die Aktion beschreibt den Wunsch oder das Ziel des OEM, die Dokumentation über OTA(Over-The-Air) Updates zu erhalten. Die Bedingung gibt an, welche Art von Updates vom OEM verantwortet werden sollen, hier alle Over-The-Air Updates. In dieser Anforderung werden keine speziellen technischen Details beschrieben. Die Komponenten zeigen, dass der Schwerpunkt auf der Verantwortung des OEM liegt eine klare Dokumentation für diese Updates in Auftrag zu geben.

Software Requirement: **„The system SW shall have a binary variable for stopping the motor when it is set to true“**	
Akteur:	„System SW“
Aktion:	„shall have a binary variable for stopping the motor“
Bedingung/ Akzeptanzkriterium:	„stopping the motor when it is set to true“
Ziel/Erwartetes Ergebnis:	Safety mechanism for motor control

Technische Details:	Binäre Variable (true/false)
Abhängigkeiten:	Motor Stop
Edge Case:	Delay in motor stop

Hier ist die Software des Systems der Akteur, welcher die Variable enthält. Diese Aktion beschreibt, dass das System eine binäre Variable für das Anhalten des Motors enthalten muss. Die Bedingung ist, dass diese binäre Variable in der Lage sein soll, den Motor anzuhalten, wenn sie auf „true“ gesetzt wird. Die Variable ist als „binary“ spezifizier, wodurch definiert wird,, dass nur zwei Zustände (true/false) unterstützt werden, und dass bei einem Wert von true der Motor gestoppt wird. In dieser Zerlegung wird deutlich, dass der Test Case überprüfen sollte, ob das System tatsächlich eine solche Variable besitzt und ob diese wie gefordert den Motor stoppt, wenn sie auf true gesetzt wird.

System Requirement**: „ The system shall be able to drive at a maximum speed of 50 cm/s“**	
Akteur:	„System“
Aktion:	„shall be able to drive“
Bedingung/ Akzeptanzkriterium:	„at a maximum speed of 50 cm/s “
Ziel/Erwartetes Ergebnis:	Safe vehicle movement
Technische Details:	Maximum speed: 50 cm/s
Abhängigkeiten:	Motion Control
Edge Case:	Power not cut off in time

Das System ist der Akteur, welcher die Aktion zur Begrenzung der maximalen Geschwindigkeit ausführt. Die Aktion beschreibt, dass das System fahren können muss. Die Bedingung ist, dass die maximale Geschwindkeit beim Fahren bei 50 cm/s liegen soll. Hier ist das technische Detail die Angabe der maximalen Geschwindigkeit. Es gibt zudem eine Abhängigkeit zu einer anderen Anforderung, die beachtet werden muss. Der Test Case sollte sicherstellen, dass das System bei einer Geschwindkeit von 50 cm/s die Energiezufuhr abschaltet. Für weitere Informationen zur Aktion des Fahrens sollte die abhängige Anforderung „Motion Control“ hinzugezogen werden.

Durch die Zerlegung der Anforderungen in diese Bestandteile wird sichergestellt, dass die Test Cases spezifisch auf die einzelnen Bestandteile der Anforderungen eingehen können. Diese Komponentenorientierung erleichtert die Identifikation relevanter Testschritte und ermöglicht eine präzise und vollständige Evaluation des Tools. Ein gut strukturierter Test Case deckt alle relevanten Bestandteile ab, wodurch eine umfassende und zielgerichtete Prüfung der Anforderung gewährleistet wird. In den weiteren Kapiteln werden die Ergebnisse evaluiert. Die detaillierte Auswertung dazu ist in Anhang B zu finden.

5.2.1 Kennzahl-Evaluierung der generierten Test Cases

Die Ergebnisse der in Kapitel 4.4 vorgestellten Kennzahlen Precision, Recall und F1-Score und Konsistenz liefern wichtige Einblicke in die Leistungsfähigkeit des entwickelten Tools zur Generierung von Test Cases für Anforderungen. Im Folgenden werden die einzelnen Kennzahlen ausgewertet, interpretiert und deren Bedeutung für die Bewertung der generierten

Test Cases detailliert erläutert. Die detaillierte Auswertung ist zusätzlich in Anhang B ausgeführt.

Tabelle 5-2 Ergebnisse der Auswertung der Kennzahlen für die Ergebnisse von GPT-4o

	Precision	Recall	F1-Score	Konsistenz
Durchschnitt	0,9772	0,8873	0,9258	0,2065
Minimum	0,75	0,333	0,7273	-0,51
Maximum	1	1	1	0,67

Der durchschnittliche Precision-Wert beträgt 0,9772. Die generierten Test Cases enthalten nahezu ausschließlich relevante Komponenten. Diese hohe Präzision zeigt, dass das Tool in der Lage ist, die Anforderungen gezielt zu adressieren, ohne irrelevante oder unnötige Inhalte zu integrieren. Der minimale Precision-Wert liegt bei 0,75 und lässt in Einzelfällen darauf schließen, dass irrelevante Inhalte aufgenommen werden. Dennoch bleibt die Präzision auch in diesen Fällen auf einem akzeptablen Niveau. Der Maximalwert von 1 belegt, dass das Tool in den besten Fällen ausschließlich relevante Inhalte generiert. Insgesamt zeigt die hohe Präzision, dass das Tool eine zuverlässige Auswahl der Anforderungen trifft und keine unnötigen Zusatzinformationen einfügt.

Der durchschnittliche Recall-Wert beträgt 0,8873 und verdeutlicht, dass das Tool die meisten relevanten Komponenten der Anforderungen in den generierten Test Cases abdeckt. Dies weist auf eine hohe Abdeckung hin, auch wenn der Wert nicht vollständig ist. Der minimale Recall-Wert von 0,333 zeigt jedoch, dass es in einzelnen Fällen zu einer unzureichenden Berücksichtigung relevanter Anforderungen kommt. Der Test Case der Anforderung „Mechanical Movement“, die auch schon in der Einleitung von Kapitel 5.2 beschrieben wurde wies beispielsweise einige Fehler auf. Der Test Case ist in Tabelle 5-3 abgebildet. Im Test Case wird zwar das System als Akteur aufgenommen, und in Bewegung gesetzt, jedoch werden sonst keine weiteren Komponenten der Anforderung berücksichtigt.

Tabelle 5-3 Beispiel eines fehlerhaften Test Cases

Anforderung	Mechanical Movement: The system shall be able to drive at a maximum speed of 50 cm/s
Test Case	All systems should initialize without errors; indicator lights should show that the actuator is ready. The actuator should begin moving towards position X; movement sound should be heard if audible, and an indicator should show the state of motion (e.g., moving). The position sensor/encoder should indicate that the actuator reaches position X within the specified tolerance (±) The actuator moves away from position X and proceeds to position Y, again indicated by the control interface. The position sensor/encoder should confirm that the actuator has reached position Y and stop movement within the specified tolerance. The time duration should be within the expected response time as per specifications (for example, 2 seconds). The actuator should return to the initial position without overshooting, again being verified by position feedback. The system should shut down without any alarms or failure states.

Dies könnte auf Schwächen bei der Erkennung oder Interpretation impliziter Anforderungen hindeuten. Die Edge Cases werden nur in wenigen Fällen berücksichtigt. Zur Verbesserung des Prompts sollte eine neue Version explizit die Beschreibung des Edge Cases beinhalten. Gleichzeitig zeigt der Maximalwert von 1, dass in den besten Fällen alle relevanten Aspekte der Anforderungen vollständig berücksichtigt werden. Der Recall-Wert verdeutlicht somit, dass das Tool in der Regel eine sehr gute Abdeckung erreicht, aber in spezifischen Fällen Optimierungspotenzial besteht.

Der durchschnittliche F1-Score von 0,9258 weist auf eine hohe Gesamtqualität der generierten Test Cases hin. Da jedoch die uneingeschränkte Anwendbarkeit der Test Cases gewährleistet sein muss, wofür ein F1-Score von 1 erforderlich wäre, bleibt weiterhin Optimierungspotenzial bestehen. Der F1-Score kombiniert Precision und Recall und zeigt somit, dass das Tool eine weitestgehend ausgewogene Balance zwischen Präzision und Vollständigkeit erreicht. Ein minimaler F1-Score von 0,7273 deutet darauf hin, dass in einigen Fällen entweder die Präzision oder die Abdeckung nicht optimal ist. Der Maximalwert von 1 zeigt jedoch, dass das Tool in den besten Fällen sowohl eine perfekte Präzision als auch eine vollständige Abdeckung der Anforderungen erzielt. Der Durchschnittswert des F1-Scores bestätigt, dass das Tool in der Regel leistungsfähig ist und qualitativ hochwertige Test Cases generiert.

Zusammenfassend zeigen die Ergebnisse, dass das Tool eine noch nicht ganz akzeptable Genauigkeit bei der Generierung der Test Cases erreicht, obwohl in den meisten Fällen eine nahezu vollständige Abdeckung der Anforderungen gewährleistet wird. Der F1-Score zeigt die gute Balance zwischen diesen beiden Aspekten und legt damit die Effizienz des Tools nahe. Dennoch weisen die minimalen Werte der Kennzahlen darauf hin, dass in Einzelfällen Optimierungspotenzial besteht, insbesondere hinsichtlich der Abdeckung aller relevanten Anforderungen. Zukünftige Verbesserungen könnten sich auf die Analyse dieser Fälle konzentrieren, um das Tool weiter zu verfeinern und die Konsistenz in der Abdeckung und Präzision zu erhöhen. Die Ergebnisse bestätigen insgesamt, dass das Tool eine vielversprechende Grundlage für die automatisierte Generierung von Test Cases darstellt und in der Praxis effektiv einsetzbar ist.

Um die Konsistenz der Test Cases auszuwerten, werden Funktionen aus den Python-Bibliotheken Pandas und Scikit-learn verwendet. Um die Ähnlichkeit der Anforderungen zu messen, werden die Test Cases mithilfe von TF-IDF (Scikit-learn) in numerische Vektoren umgewandelt. Dieser Ansatz ermöglicht es, die semantische Ähnlichkeit zwischen Anforderungen mithilfe der Cosine Similarity (Scikit-learn) zu berechnen, die Werte im Bereich von -1 (gegenteilig) bis 1 (identisch) liefert. Ein Schwellenwert von 0,5 wird festgelegt, um Anforderungen zu identifizieren, die als ähnlich genug gelten, um in die Konsistenzanalyse einbezogen zu werden. Für jede Anforderung, deren Ähnlichkeit den Schwellenwert überschreitet, wird die Konsistenz der zugehörigen Test Cases berechnet. Diese setzt sich aus den zwei in Kapitel 3.4 erläuterten Komponenten zusammen: der semantischen Ähnlichkeit der Test Case-Beschreibungen und der strukturellen Konsistenz. Die semantische Ähnlichkeit der Beschreibungen wird ebenfalls mit TF-IDF und der Cosine Similarity ermittelt. Die strukturelle Konsistenz wird anhand der Anzahl der Schritte in den Test Cases berechnet. Hierbei wird berücksichtigt, wie stark sich die Schrittanzahl unterscheidet, und die Übereinstimmung der relativen Schrittanzahl berechnet. Die Konsistenz ergibt sich schließlich

als Differenz zwischen der semantischen Ähnlichkeit der Beschreibungen und der strukturellen Konsistenz.

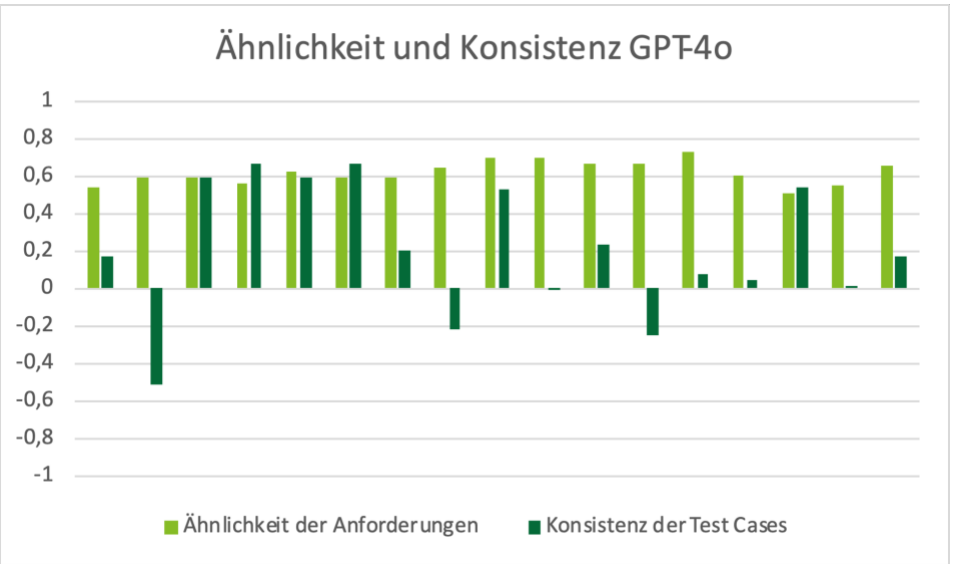


Abbildung 5-6 Ähnlichkeit und Konsistenz der generierten Test Cases von GPT-4o

Die berechneten Konsistenzwerte für Test Cases, die auf Anforderungen mit einer Ähnlichkeit von mindestens 0,5 basieren, liegen in einem Bereich von -0,51 bis 0,67. Positive Werte zeigen, dass die Test Cases in Bezug auf ihre Struktur und ihre Beschreibungen konsistent sind, während negative Werte auf Diskrepanzen hinweisen. Eine solche Verteilung verdeutlicht, dass die Konsistenz sowohl von der strukturellen Ausgestaltung der Test Cases als auch von der semantischen Übereinstimmung der Beschreibungen beeinflusst wird. Die positiven Konsistenzwerte (z. B. 0,67, 0,59 und 0,53) verdeutlichen, dass die Test Cases sowohl strukturell als auch semantisch übereinstimmen. Dies weist darauf hin, dass für diese Test Cases eine ähnliche Anzahl von Schritten sowie vergleichbare Formulierungen in den Beschreibungen vorliegen. Diese Übereinstimmung spricht für eine harmonisierte Test Case-Entwicklung, bei der Anforderungen mit ähnlichem Inhalt konsistente Test Cases nach sich ziehen. Im Gegensatz dazu deuten die negativen Werte (z. B. -0,51 und -0,22) auf Inkonsistenzen hin.

In Tabelle 5-4 sind zwei ähnliche Anforderungen aufgeführt, dessen Test Cases sich im Vergleich nicht so stark ähneln. Die Anzahl der Schritte bleibt gleich, jedoch sind Schritte sehr unterschiedlich formuliert. Auch der Inhalt unterscheidet sich mehrfach, wie schon in Schritt 2 zu erkennen ist.

Tabelle 5-4 Beispiel von inkonsistenten Test Cases von GPT-4o

Anforderung	Test Case
Yaw Rate - Speed: The system SW shall have a faster rotation movement when the yaw rate's absolute value is bigger	1. Action Set the yaw rate to a value below the predefined threshold (e.g., 5 degrees/second). 2. Action Gradually increase the yaw rate to the threshold value (e.g., 10 degrees/second) while observing the system's response. 3. Action Set the yaw rate to a value significantly above the threshold (e.g., 20 degrees/second). 4. Action Vary the yaw rate rapidly between lower and higher values (e.g., 5 degrees/second to 25 degrees/second in a staggered manner).

	5. Action Reset the yaw rate to zero and monitor the system’s response. The system should show normal rotation speed (baseline speed) on the movement display. Once the threshold is crossed, the system should indicate a change in rotation speed (indicating an initial increase in rotation). The system should demonstrate a noticeably faster rotation movement than the baseline speed and should maintain that speed consistently. The system should respond swiftly to the changes in yaw rate, accelerating and decelerating as appropriate, maintaining the relationship between yaw rate and rotation speed. The rotation should cease or revert to a stationary state as the yaw rate returns to zero.
Yaw Rate - Counterclockwise: The system SW shall have a counterclockwise rotation when the yaw rate <0	1. Action Simulate a negative yaw rate (e.g., -30 degrees per second). 2. Action Monitor the system's response upon detecting the negative yaw rate. 3. Action Observe the motor or the system's visual indicators to confirm rotation direction. 4. Action Measure the actual rotation using a protractor or motion capture, if applicable. 5. Action Gradually decrease the absolute value of the negative yaw rate to see if the rotation slows down appropriately. The system should register the yaw rate as less than zero. The motor control system should activate the counterclockwise rotation command. The system physically rotates counterclockwise, and visual indicators (if any) reflect the change in motion. The rotation should show a consistent counterclockwise motion as per the specified yaw rate input. The system should correctly reduce the speed of counterclockwise rotation in accordance with the changing yaw rate.

Das zeigt, dass auch bei sehr ähnlichen Anforderungen noch teilweise sprachliche Unstimmigkeiten vorliegen. Die durchschnittliche Konsistenz von 0,20 zeigt, dass die Test Cases im Mittel eher konsistent sind. Der Median der Auswertung liegt bei 0,17. Dies bedeutet, dass die semantische Ähnlichkeit leicht stärker ins Gewicht fällt als die strukturelle Konsistenz (z. B. die Übereinstimmung in der Anzahl der Schritte) der Test Case-Beschreibungen. Solche Werte nahe Null deuten darauf hin, dass keine gravierenden Probleme vorliegen, die Konsistenz jedoch durch eine stärkere Harmonisierung der Test Case-Strukturen und -Beschreibungen verbessert werden könnte. Insgesamt zeigt die Verteilung der Werte, dass die Mehrzahl der Test Cases relativ konsistent ist, da die meisten Werte im positiven oder leicht negativen Bereich liegen. Dennoch gibt es vereinzelte Test Cases, bei denen erhebliche

Inkonsistenzen festgestellt werden. Diese sollten im Rahmen einer detaillierten Analyse näher betrachtet werden, um die Ursachen der Abweichungen zu identifizieren und gegebenenfalls zu beheben.

5.2.2 Kennzahl-Evaluierung der Benchmark Test Cases

Zusätzlich zu den generierten Test Cases aus dem vorgestellten Tool werden auch Test Cases mit dem spezialisierten Tool generiert. Der Prozess dafür unterscheidet sich stark von dem in dieser Arbeit vorgestellten Prozess. Dadurch, dass dieses Tool direkt in Codebeamer integriert ist, muss weder ein Export noch ein Import stattfinden. Das Tool besitzt einige Fähigkeiten, die über die Test Case Generierung hinaus gehen. Die Test Case Generierung findet hierbei über den „Test Planner“ statt, der zunächst einige Vorschläge für Test Case Konditionen macht. Das Tool analysiert den Anforderungstext und stellt verschiedene Formulierungen für Test Case Beschreibungen zur Verfügung aus denen eine oder mehrere ausgewählt werden können, die dann zu einem neuen Test Case Item erstellt werden. Zur Befüllung der einzelnen Felder dieses Test Cases muss dann der integrierte Chatbot hinzugezogen werden, mit Hilfe dessen der Requirements Engineer die Pre-Action, Post-Actions, Test Parameters, Test Steps und Remarks erarbeiten kann. Dies nimmt mehr Zeit in Anspruch als die Batch-Generierung, die in dieser Arbeit vorgestellt wurde, kann aber auch durch die Einbindung des Experten insgesamt bessere Ergebnisse liefern. Um hier die Vergleichbarkeit der Generierungsfähigkeiten der Tools zu gewährleisten, werden allein Vorschläge des Chatbots in den Test Case einbezogen, ähnlich zu dem Prompt der vorher an GPT-4o übergeben wurde. Der Prompt für die Erstellung der Test Steps ist für diese Auswertung der gleiche wie in Kapitel 4 beschrieben. Für die Evaluierung werden in beiden Fällen nur die Test Steps (Actions und Expected Results) betrachtet. Die Ergebnisse der Auswertung der Kennzahlen sind in Tabelle 5-5 zu sehen.

Tabelle 5-5 Ergebnisse der Auswertung der Kennzahlen für die Ergebnisse des spezialisierten Tools

	Precision	Recall	F1-Score	Konsistenz
Durchschnitt	0,8629	0,8730	0,8665	0,1233
Min	0,5	0,5714	0,5714	-0,51
Max	1	1	1	0,67

Der durchschnittliche Precision-Wert beträgt 0,8629 und zeigt, dass die generierten Test Cases in der Regel einen hohen Anteil relevanter Komponenten enthalten. Dies deutet darauf hin, dass das Tool in der Lage ist, irrelevante oder überflüssige Inhalte weitgehend zu vermeiden. Der minimale Precision-Wert von 0,5 offenbart jedoch, dass es in einzelnen Fällen zu einer erheblichen Aufnahme irrelevanter Komponenten kommt. Dies könnte darauf hindeuten, dass das Tool bei der Erkennung der Relevanz von Komponenten Schwächen aufweist, insbesondere bei komplexeren Anforderungen. Der Maximalwert von 1 zeigt, dass das Tool in bestimmten Fällen perfekt präzise arbeitet und ausschließlich relevante Inhalte generiert. Insgesamt zeigt die durchschnittliche Precision, dass die generierten Test Cases weitgehend präzise sind, aber Optimierungspotenzial in der Vermeidung irrelevanter Inhalte besteht.

Der durchschnittliche Recall-Wert beträgt 0,8730 und liegt damit knapp über der Precision. Dies deutet darauf hin, dass das Tool die meisten relevanten Komponenten der Anforderungen berücksichtigt und weitestgehend eine gute Abdeckung erzielt. Der minimale Recall-Wert von 0,5714 zeigt jedoch, dass in bestimmten Fällen relevante Komponenten der Anforderungen nicht vollständig in den Test Cases enthalten sind. Dies weist darauf hin, dass die Fähigkeit des Tools, alle relevanten Aspekte komplexer Anforderungen zu identifizieren und abzudecken, noch Verbesserungspotential besitzt. Der Maximalwert von 1 belegt, dass das Tool in den besten Fällen eine vollständige Abdeckung der Anforderungen erreicht. Der

durchschnittliche Recall-Wert unterstreicht, dass das Tool in der Regel eine solide Abdeckung bietet, aber in spezifischen Fällen noch Verbesserungspotenzial hat.

Der durchschnittliche F1-Score beträgt 0,8665 und zeigt auch hier ein noch nicht ganz akzeptables Gleichgewicht zwischen Präzision und Vollständigkeit. Der minimale F1-Score von 0,5714 zeigt zudem, dass in den schlechtesten Fällen weder eine ausreichende Präzision noch eine vollständige Abdeckung erreicht werden. Dies könnte darauf hinweisen, dass das Tool bei komplexen oder weniger klar definierten Anforderungen Schwierigkeiten hat, ein optimales Ergebnis zu liefern. Der Maximalwert von 1 zeigt, dass das Tool in den besten Fällen sowohl präzise als auch vollständig arbeitet. Der durchschnittliche F1-Score zeigt, dass das Tool in der Regel qualitativ hochwertige Test Cases generiert, auch wenn in Einzelfällen Schwächen auftreten können.

Zusammenfassend lässt sich feststellen, dass das spezialisierte Tool Test Cases mit einer insgesamt guten Qualität generiert, wobei der durchschnittliche Recall leicht höher ist als die Precision und weist damit auf eine stärkere Fokussierung auf die Abdeckung der Anforderungen hin. Der F1-Score bestätigt, dass das Tool ein ausgewogenes Verhältnis zwischen Präzision und Vollständigkeit erreicht. Die minimalen Werte der Kennzahlen zeigen jedoch, dass es in bestimmten Fällen zu deutlichen Schwächen kommt, sowohl hinsichtlich der Präzision als auch der Abdeckung. Diese Schwächen könnten auf Herausforderungen bei der Verarbeitung komplexer oder impliziter Anforderungen zurückzuführen sein. Im Vergleich zu den durchschnittlichen Werten zeigt das Tool eine solide Leistung, es besteht jedoch Potenzial zur Verbesserung der Konsistenz und der Fähigkeit, in allen Fällen präzise und vollständige Test Cases zu generieren.

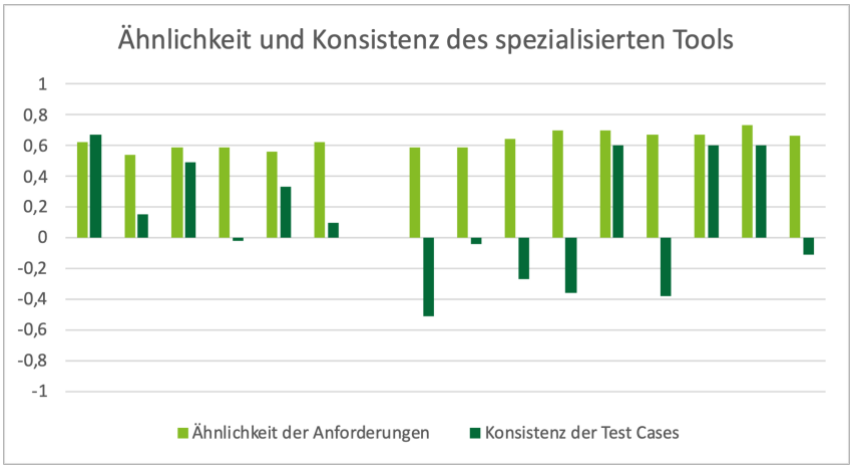


Abbildung 5-7 Ähnlichkeit und Konsistenz der generierten Test Cases des spezialisierten Tools

Die Ergebnisse der Analyse der Konsistenz zeigen, dass Paare mit hoher semantischer Ähnlichkeit der Anforderungen tendenziell eine positive Konsistenz der Test Cases aufweisen. Beispielsweise erreichen die Anforderungen "Directional Movement - Front" und "Directional Movement - Back" eine semantische Ähnlichkeit von 0,73 und eine Konsistenz der Test Cases von 0,60. Dieses Ergebnis deutet darauf hin, dass das Modell ähnliche Anforderungen konsistent verarbeitet und eine einheitliche Struktur der Test Cases erzeugt. Um zusätzlich zum Negativbeispiel in Tabelle 5-4 in Kapitel 5.2.1 ein Positivbeispiel anzugeben, sind die genauen Anforderungen sowie Test Cases sind in Tabelle 5-6 dargestellt. In beiden Test Cases wird zunächst die Bewegungsrichtung festgelegt, wie in der Anforderung festgelegt. Anschließend wird das System aktiviert und in Bewegung gesetzt. Dann wird das gewünschte Ergebnis verifiziert bzw. mögliche Fehler dokumentiert.

Die Schwierigkeiten des Tools zeigen sich, ähnlich wie in der Auswertung in Kapitel 5.2.1 bei der Verarbeitung der „Yaw Rate“-Anforderungen. Trotz einer Ähnlichkeit von 0,59 zwischen den Anforderungen weisen die Inhalte deutliche Unterschiede auf. Diese Ergebnisse verdeutlichen, dass strukturelle Unterschiede in den Test Cases stärker gewichtet werden, wenn die Anforderungen unterschiedliche Rahmenbedingungen adressieren.

Tabelle 5-6 Beispiel von konsistenten Test Cases des spezialisierten Tools

Anforderung	Test Case
Directional Movement - Left: The system SW shall have a left directional movement when the movement direction parameter corresponds is 180°	This test case verifies the left directional movement of the system SW when the movement direction parameter is 180° and the system is in a functional state. 1. Set the movement direction parameter to 180°. 2. Initiate the system SW. 3. Observe the direction of movement. 4. Expected Result: The system SW should move to the left. 5. Document the observation and any errors encountered.
Directional Movement - Front: The system SW shall have a forward directional movement when the movement direction parameter corresponds is 90°	This test case checks that the system SW does not move forward when the movement direction parameter is 90° and the system is in the operational state. 1. Set the movement direction parameter to 90°. 2. Place the system in a operational state. 3. Attempt to activate the system SW. 4. Verify that the system does move forward. 5. Document the results and any errors encountered.

Ein weiteres Muster zeigt sich bei Anforderungen, die eng miteinander verwandt sind, jedoch verschiedene Aspekte derselben Funktionalität adressieren. Beispielsweise weisen die Anforderungen "Collision Avoidance Distance" und "Collision Avoidance" eine semantische Ähnlichkeit von 0,62 und eine Konsistenz von 0,67 auf. Dies zeigt, dass das Modell in der Lage ist, die Konsistenz für eng verwandte Anforderungen aufrechtzuerhalten, indem es sowohl semantische als auch strukturelle Aspekte berücksichtigt.

Die Analyse der Konsistenz-Kennzahl zeigt insgesamt, dass semantische Ähnlichkeit allein nicht ausreicht, um die Konsistenz der Test Cases zu gewährleisten. Strukturelle Unterschiede, wie sie durch die Anzahl und Art der Schritte in den Test Cases erfasst werden, spielen eine zentrale Rolle. Hohe Konsistenzwerte bei ähnlichen Anforderungen deuten darauf hin, dass das Modell in diesen Fällen die Anforderungen kohärent und mit einem klaren strukturellen Schema verarbeitet. Negative Konsistenzwerte bei inhaltlich verwandten Anforderungen legen jedoch Optimierungspotenzial im Generierungsprozess offen, insbesondere in Bezug auf die Berücksichtigung struktureller Rahmenbedingungen. Auch diese Evaluation ist in ausführlicher Form in Anhang B vorhanden.

5.2.3 Vergleich des entwickelten Tools und Benchmark

Die Ergebnisse der Performanceanalyse beider Large Language Models zeigen unterschiedliche Schwerpunkte in ihrer Leistungsfähigkeit bei der Generierung von Test Cases im Bereich Requirements Engineering. Hinter dem ersten Tool steht ein allgemein entwickeltes, jedoch leistungsstarkes LLM (GPT-4o), während das zweite Tool auf ein spezialisiertes Modell zurückgreift, das explizit für Anwendungen im Requirements Engineering entwickelt wurde.

Das Ergebnis von GPT-4o erreicht eine durchschnittliche Precision von 0,9772, während das spezialisierte LLM 0,8629 erzielt. Dieser Unterschied deutet darauf hin, dass das allgemeine Modell relevantere Test Cases generiert, da weniger irrelevante oder überflüssige Inhalte enthalten sind. Der minimale Precision-Wert des spezialisierten Modells liegt mit 0,5 deutlich

unter dem des allgemeinen Modells, welches einen Minimalwert von 0,75 aufweist. Beide Modelle erzielen jedoch einen Maximalwert von 1, sie können beide unter optimalen Bedingungen perfekt präzise arbeiten.

Im Recall sind die Modelle vergleichbarer. GPT-4o erreicht einen Durchschnittswert von 0,8873, während das spezialisierte Modell mit 0,8730 nur geringfügig darunter liegt. Dies zeigt, dass beide Modelle die relevanten Anforderungen in den meisten Fällen gut abdecken. Der minimale Recall-Wert ist bei GPT-4o niedriger und liegt bei 0,333. Dies weist auf mögliche Schwächen bei der Abdeckung relevanter Komponenten in spezifischen Fällen hin. Der Maximalwert von 1 verdeutlicht jedoch, dass beide Modelle unter idealen Bedingungen eine vollständige Abdeckung gewährleisten können.

Der F1-Score, der als harmonisches Mittel von Precision und Recall eine Gesamtbewertung der Modelle ermöglicht, beträgt für das allgemeine LLM 0,9258, während das spezialisierte Modell 0,8665 erreicht. Dieser Unterschied zeigt, dass GPT-4o eine bessere Balance zwischen Präzision und Vollständigkeit bietet. Die minimalen F1-Scores zeigen ebenfalls Unterschiede: GPT-4o erreicht 0,7273, während das spezialisierte Modell mit 0,5714 in den schlechtesten Fällen schwächere Ergebnisse liefert.

Die Analyse zeigt, dass das allgemeine LLM bei der Generierung von Test Cases eine höhere Präzision und eine insgesamt bessere Balance zwischen Präzision und Vollständigkeit aufweist. Dies könnte auf die breitere Datenbasis und die größere Modellkapazität des allgemeinen Modells zurückzuführen sein, die es ermöglichen, auch in spezialisierten Bereichen wie dem Requirements Engineering leistungsstark zu agieren. Das spezialisierte LLM zeigt hingegen eine gute, aber weniger konsistente Leistung, insbesondere bei der Präzision und der Balance zwischen Abdeckung und Relevanz. Die Ergebnisse legen nahe, dass das allgemeine LLM für Anwendungen geeignet ist, bei denen einfache, aber genaue Anforderungen vorliegen, die vom LLM effizient zu Test Cases verarbeitet werden können. Das spezialisierte LLM könnte durch gezielte Optimierungen und Erweiterungen seiner Trainingsdaten oder Architektur weiter verbessert werden, um eine konsistentere Leistung zu erzielen und seine Spezialisierung im Requirements Engineering besser auszuschöpfen. Durch due direkte Einbindung im ALM Tool, ist die Verwendung als Assistenz zur Erstellung von komplexen Test Cases gut geeignet.

Der Vergleich der Konsistenzwerte der beiden Tools - des allgemeinen LLMs GPT-4o und des spezialisierten Tools - zeigt geringe Unterschiede in ihrer Fähigkeit, strukturelle und semantische Übereinstimmung zwischen Test Cases und den zugrunde liegenden Anforderungen zu gewährleisten. Die Konsistenzwerte der Test Cases wurden auf Basis der strukturellen Konsistenz und der semantischen Ähnlichkeit berechnet und reichen bei beiden Tools über einen breiten Wertebereich, mit exakt den gleichen Extremwerten. GPT-4o erreicht Konsistenzwerte im Bereich von -0,51 bis 0,67, während das spezialisierte Tool Konsistenzwerte zwischen -0,51 und 0,67 aufweist. Dies zeigt, dass beide in optimalen Fällen in der Lage sind, eine hohe Konsistenz zwischen den Test Cases und den Anforderungen zu erzielen. Allerdings weist das spezialisierte Tool häufiger negative Werte auf, wodurch ein leicht niedriger Durchschnittswert erzielt wird.

Die Häufigkeit positiver Konsistenzwerte ist bei GPT-4o höher deutet darauf hin, dass es in mehr Fällen eine zufriedenstellende Übereinstimmung zwischen Struktur und Inhalt der Test Cases erreicht. Insgesamt konnten hier aber keine Ergebnisse erzielt werden, die sofort in der Industrie Anwendung finden könnten. Das spezialisierte Tool zeigt eine größere Streuung der Ergebnisse, mit einigen Werten im negativen Bereich und weist auf eine geringere Stabilität und Konsistenz hin. Dieses Ergebnis könnte darauf zurückzuführen sein, dass das spezialisierte Tool stärker auf die semantische Nähe der Anforderungen fokussiert ist und weniger Gewicht auf die Harmonisierung der Test Case-Struktur legt.

Zusammenfassend lässt sich feststellen, dass GPT-4o in Bezug auf die Konsistenz der Test Cases eine etwas gleichmäßigere und stabilere Leistung zeigt. Das spezialisierte Tool erzielt in einigen Fällen ähnliche maximale Konsistenzwerte, weist jedoch etwas häufiger negative Werte auf, die auf eine leicht stärkere Varianz und potenzielle Schwächen in der strukturellen

Ausgestaltung der Test Cases hindeuten. Die Ähnlichkeit und Höhe der Werte deuten darauf hin, dass beide Tools noch Optimierungspotential bei der Generierung von konsistenten Test Cases aufweisen.

5.3 Diskussion und Fazit der Ergebnisse der Integration und Evaluierung

Die Evaluation der beiden Tools - GPT-4o als allgemeines LLM und das spezialisierte Tool - zeigt nur leichte Unterschiede in ihrer Leistungsfähigkeit bei der Generierung von Test Cases. Beide Modelle weisen spezifische Stärken und Schwächen auf, die sich sowohl in den quantitativen Kennzahlen als auch in qualitativen Aspekten der generierten Test Cases widerspiegeln. Die Evaluation zeigt, je präziser die Anforderungen, desto präziser werden auch die Test Cases formuliert. Bei Anforderungen ohne Interpretationsspielraum werden auch sehr präzise Test Cases erstellt, je weniger granular die Anforderungen jedoch sind, desto schlechter wirkt sich das auf die Qualität der Test Cases aus. Beide Modelle haben Schwierigkeiten, unklare oder fehlende Details sinnvoll zu ergänzen. Dies verdeutlicht, dass die Qualität der Eingabedaten einen entscheidenden Einfluss auf die Leistungsfähigkeit der Modelle hat. Darüber hinaus wurde deutlich, dass Edge Cases und mathematische Berechnungen Herausforderungen darstellen, die beide Tools noch nicht vollständig bewältigen. Während Edge Cases in der Regel nicht automatisch berücksichtigt werden, könnten sie durch eine gezielte Anpassung der Prompts leichter in die Test Case-Generierung integriert werden. Mathematische Anforderungen hingegen erfordern eine umfassendere Weiterentwicklung der Modelle, um eine präzisere Verarbeitung solcher Aufgaben zu gewährleisten.

Die Konsistenz der generierten Test Cases variiert kaum zwischen den Tools. Beide LLMs zeigen eine Leistung mit Optimierungspotential zwischen semantischer und struktureller Konsistenz. Besonders das Auftreten von Abweichungen der Test Cases bei Anforderungen mit inhaltlicher Ähnlichkeit zeigen die Defizite auf, die auf ein unzureichendes Gleichgewicht zwischen semantischen und strukturellen Aspekten hinweisen. Die Einbindung zusätzlicher Datenquellen, wie etwa Feedback aus früheren Entwicklungsprojekten oder historischen Test Case-Daten, könnte die Modellleistung weiter verbessern.

Insgesamt zeigt sich, dass beide Tools unterschiedliche Stärken und Schwächen haben. GPT-4o ist besonders geeignet für standardisierte Szenarien mit klaren und gut strukturierten Anforderungen, da es konsistente und präzise Ergebnisse liefert. Das spezialisierte Tool hingegen bietet durch seine direkte Integration in bestehende Systeme und die Möglichkeit zur interaktiven Anpassung der Test Cases Vorteile in komplexeren und flexibleren Anwendungsfällen. Seine Stärken liegen in der semantischen Verarbeitung von Anforderungen, während es bei der strukturellen Konsistenz Verbesserungsbedarf aufweist.

Die Evaluation unterstreicht, dass die Wahl des geeigneten Tools stark vom Anwendungsfall abhängt. Für Szenarien mit klar definierten Anforderungen und hohen Anforderungen an Präzision und Konsistenz bietet GPT-4o Vorteile. Das spezialisierte Tool eignet sich besser für dynamische und variable Anforderungen, bei denen die Einbindung von Experten und die Anpassungsfähigkeit des Tools eine größere Rolle spielen. Beide Tools könnten durch gezielte Weiterentwicklungen weiter verbessert werden. Insbesondere die Berücksichtigung von Edge Cases und die Fähigkeit, mathematische Berechnungen in Test Cases zu integrieren, sind wichtige Optimierungsbereiche. Die Ergebnisse zeigen, dass die Automatisierung von Test Case Generierungen mithilfe von LLMs bereits jetzt eine vielversprechende Grundlage bietet, jedoch von der Qualität der Anforderungen und den spezifischen Stärken und Schwächen der eingesetzten Tools abhängig ist.

6 Zusammenfassung und Ausblick

Diese Arbeit untersucht die Integration von Natural Language Processing in Application Lifecycle Management und liefert sowohl theoretische als auch praktische Einblicke in die Möglichkeiten und Herausforderungen dieses Ansatzes. Mithilfe des CRISP-DM-Modells wird ein konzeptioneller Rahmen entwickelt, der die schrittweise Implementierung von NLP in ALM-Systeme ermöglicht. Neben einer umfassenden Literaturrecherche zu bestehenden Ansätzen werden konkrete Anwendungsfälle evaluiert, wobei ein besonderer Fokus auf die Automatisierung der Generierung von Test Cases gelegt wird.

Die Ergebnisse zeigen, dass NLP das Potenzial besitzt, sowohl die Effizienz als auch die Qualität in ALM-Prozessen erheblich zu steigern. Durch die Automatisierung von Routinetätigkeiten wie der Anforderungsklassifikation, der Konsistenzprüfung und der Test Case-Generierung können Ressourcen effizienter genutzt und Fehler reduziert werden. Dennoch wird festgestellt, dass die Qualität der Ergebnisse stark von der Präzision und Granularität der zugrunde liegenden Anforderungen abhängt. Anforderungen, die klar definiert und wenig interpretationsbedürftig sind, führen zu präziseren und vollständigeren Test Cases. Bei weniger klaren Anforderungen stößt das NLP jedoch an Grenzen, insbesondere bei der Berücksichtigung von Edge Cases und mathematischen Berechnungen. Es ist zu beachten, dass die Qualität von generierten Test Cases stark von der Qualität der vorausgehenden Anforderungen abhängt.

Der Vergleich eines allgemeinen LLM mit einem spezialisierten Modell verdeutlicht, dass ein spezialisierter Ansatz nicht automatisch zu besseren Ergebnissen führt. Während das allgemeine LLM höhere Konsistenz und Präzision in der Generierung von Test Cases aufweist, kann das spezialisierte Modell in bestimmten Anwendungsfällen durch domänenspezifisches Wissen punkten. Diese Ergebnisse betonen die Wichtigkeit einer fundierten Auswahl und Anpassung von NLP-Modellen an die spezifischen Anforderungen des jeweiligen Anwendungsfeldes.

Die in dieser Arbeit gewonnenen Erkenntnisse bieten eine Grundlage für zukünftige Forschungen und praktische Anwendungen im Bereich der NLP-Integration in ALM. Zukünftige Arbeiten können darauf abzielen, die erkannten Schwächen der Modelle zu adressieren, insbesondere in Bezug auf die Berücksichtigung komplexer und impliziter Anforderungen. Darüber hinaus können weiterführende Untersuchungen zur Optimierung der Prompt-Strategien beitragen, um eine bessere Abdeckung von Edge Cases zu gewährleisten. Ein weiterer wichtiger Aspekt zukünftiger Forschung ist die Skalierbarkeit der entwickelten Methoden. Während die Fallstudie in dieser Arbeit auf spezifische Datensätze beschränkt ist, wäre eine Anwendung auf größere und diversifiziertere Datensätze ein entscheidender Schritt, um die Übertragbarkeit und Robustheit der Ergebnisse zu validieren.

Schließlich könnten praxisorientierte Leitfäden und Best Practices entwickelt werden, um Unternehmen bei der erfolgreichen Integration von NLP in ihre ALM-Prozesse zu unterstützen. Die kontinuierliche Weiterentwicklung von NLP-Technologien sowie die zunehmende Verfügbarkeit von hochwertigen Daten bieten vielversprechende Möglichkeiten, die Potenziale dieser Ansätze weiter auszuschöpfen und so einen bedeutenden Beitrag zur Effizienzsteigerung und Innovation im Lifecycle Management zu leisten.

7 Literaturverzeichnis

Alhoshan, Waad; Ferrari, Alessio; Zhao, Liping (2023): Zero-shot learning for requirements classification: An exploratory study. In: *Information and Software Technology* 159, S. 107202. DOI: 10.1016/j.infsof.2023.107202.

Amazon Web Services (2024): What is ALM (Application Lifecycle Management)? Online verfügbar unter https://aws.amazon.com/what-is/application-lifecycle-management/, zuletzt geprüft am 05.07.2024.

Anthropic (2024): Meet Claude. Hg. v. Anthropic. Online verfügbar unter https://www.anthropic.com/claude, zuletzt geprüft am 23.09.2024.

Arulmohan, Sathurshan; Meurs, Marie-Jean; Mosser, Sébastien (2023): Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). Västerås, Sweden, 01.10.2023 - 06.10.2023: IEEE, S. 580–587.

Atlassian (2024): What is PLM (Product Lifecycle Management)? Online verfügbar unter https://www.atlassian.com/agile/product-management/plm, zuletzt geprüft am 05.07.2024.

Bernard, Alain; Bouras, Abdelaziz; Foufou, Sebti; Ríos, José (Hg.) (2017): Product Lifecycle Management and the Industry of the Future. 14th IFIP WG 5.1 International Conference, PLM 2017, Seville, Spain, July 10-12, 2017, Revised Selected Papers. 1st ed. 2017. Cham: Springer International Publishing; Imprint: Springer (IFIP Advances in Information and Communication Technology, 517).

Bertram, Vincent; Kausch, Hendrik; Kusmenko, Evgeny; Nqiri, Haron; Rumpe, Bernhard; Venhoff, Constantin (2023): Leveraging Natural Language Processing for a Consistency Checking Toolchain of Automotive Requirements. In: 2023 IEEE 31st International Requirements Engineering Conference (RE). 2023 IEEE 31st International Requirements Engineering Conference (RE). Hannover, Germany, 04.09.2023 - 08.09.2023: IEEE, S. 212–222.

Bharadiya, Jasmin Praful (2023): A Comprehensive Survey of Deep Learning Techniques Natural Language Processing.

Bhatia, Jaspreet; Breaux, Travis D.; Schaub, Florian (2016): Mining Privacy Goals from Privacy Policies Using Hybridized Task Recomposition. In: *ACM Trans. Softw. Eng. Methodol.* 25 (3), Artikel 22, S. 1–24. DOI: 10.1145/2907942.

Bilenko, Misha (2024): Phi-3 Neue Maßstäbe für die Möglichkeiten kleiner Sprachmodelle. Hg. v. Microsoft. Online verfügbar unter https://news.microsoft.com/de-de/phi-3-neue-massstaebe-fuer-die-moeglichkeiten-kleiner-sprachmodelle/, zuletzt geprüft am 23.09.2024.

Boukhlif, Mohamed; Hanine, Mohamed; Kharmoum, Nassim; Ruigómez Noriega, Atenea; García Obeso, David; Ashraf, Imran (2024): Natural Language Processing-Based Software Testing: A Systematic Literature Review. In: *IEEE Access* 12, S. 79383–79400. DOI: 10.1109/ACCESS.2024.3407753.

Bouras, Abdelaziz; Eynard, Benoit; Foufou, Sebti; Thoben, Klaus-Dieter (Hg.) (2016): Product Lifecycle Management in the Era of Internet of Things. 12th IFIP WG 5.1 International Conference, PLM 2015, Doha, Qatar, October 19-21, 2015, Revised Selected Papers. 1st ed. 2016. Cham: Springer International Publishing; Imprint: Springer (IFIP Advances in Information and Communication Technology, 467).

Cambria, Erik; White, Bebo (2014): Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]. In: *IEEE Comput. Intell. Mag.* 9 (2), S. 48–57. DOI: 10.1109/MCI.2014.2307227.

Chapman, P.; Clinton, J.; Kerber, R.; Khabaza, T.; Reinartz, T., Shearer, C.; Wirth, R. (2000): CRISP-DM 1.0: Step-by-step data mining guide. In: *SPSS inc* 9 (13), S. 1–73.

Chappell, David (2008): What is Application Lifecycle Management? Online verfügbar unter http://www.chappellassoc.com/writing/white_papers/What-is-ALM--Chappell.pdf, zuletzt geprüft am 04.07.2024.

Chowdhury, G. G. (2003): Natural Language Processing. In: *Annual review of information science and technology* (37), S. 51–89.

Cohere (2024): Command and Command Light. Online verfügbar unter https://docs.cohere.com/docs/command-beta, zuletzt geprüft am 23.09.2024.

Couder, Juan Ortiz; Gomez, Dawson; Ochoa, Omar (2024): Requirements Verification Through the Analysis of Source Code by Large Language Models. In: SoutheastCon 2024. SoutheastCon 2024. Atlanta, GA, USA, 15.03.2024 - 24.03.2024: IEEE, S. 75–80.

Cruciani, F.; Moore, S.; Nugent, C. D. (2023): Comparing general purpose pre-trained Word and Sentence embeddings for Requirements Classification. In: *6th Workshop on Natural Language Processing for Requirements Engineering: REFSQ Co-Located Events 2023. CEUR-WS.* Online verfügbar unter https://www.scopus.com/inward/record.uri?eid=2-s2.0-85158949872&partnerID=40&md5=2d65c47a7a32f01af905fad1d647e22b.

Das, Souvick; Deb, Novarun; Cortesi, Agostino; Chaki, Nabendu (2023): Zero-shot Learning for Named Entity Recognition in Software Specification Documents. In: 2023 IEEE 31st International Requirements Engineering Conference (RE). 2023 IEEE 31st International Requirements Engineering Conference (RE). Hannover, Germany, 04.09.2023 - 08.09.2023: IEEE, S. 100–110.

Deuter, Andreas; Rizzo, Stefano (2016): A Critical View on PLM/ALM Convergence in Practice and Research. In: *Procedia Technology* 26, S. 405–412. DOI: 10.1016/j.protcy.2016.08.052.

Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina (2018): BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Online verfügbar unter http://arxiv.org/pdf/1810.04805v2.

Döring, Nicola (2023): Forschungsmethoden und Evaluation in den Sozial- und Humanwissenschaften. 6., vollständig überarbeitete, aktualisierte und erweiterte Auflage. Berlin, Heidelberg: Springer (Lehrbuch). Online verfügbar unter http://www.springer.com/.

Dos Santos, Carlos Alberto; Bouchard, Kevin; Petrillo, Fabio (2024): AI-Driven User Story Generation. In: 2024 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA). 2024 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA). Victoria, Seychelles, 01.02.2024 - 02.02.2024: IEEE, S. 1–6.

EleutherAI (2023): GPT Neo. Hg. v. EleutherAI. Online verfügbar unter https://www.eleuther.ai/artifacts/gpt-neo, zuletzt geprüft am 23.09.2024.

El-Hajjami, Abdelkarim; Fafin, Nicolas; Salinesi, Camille (2023): Which AI Technique Is Better to Classify Requirements? An Experiment with SVM, LSTM, and ChatGPT. Online verfügbar unter http://arxiv.org/pdf/2311.11547v2.

Evans, Chis; Ingerson, Brian; Ben-Kiki, Oren (2009): The Official YAML Web Site. Online verfügbar unter https://yaml.org/, zuletzt geprüft am 29.09.2024.

Explosion (2024): spaCy · Industrial-strength Natural Language Processing in Python. Online verfügbar unter https://spacy.io/, zuletzt geprüft am 23.09.2024.

Ezzini, Saad; Abualhaija, Sallam; Arora, Chetan; Sabetzadeh, Mehrdad (2023): AI-based Question Answering Assistance for Analyzing Natural-language Requirements. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). 2023 IEEE/ACM

45th International Conference on Software Engineering (ICSE). Melbourne, Australia, 14.05.2023 - 20.05.2023: IEEE, S. 1277–1289.

Fischbach, Jannik; Frattini, Julian; Vogelsang, Andreas; Mendez, Daniel; Unterkalmsteiner, Michael; Wehrle, Andreas et al. (2023): Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study. In: *Journal of Systems and Software* 197, S. 111549. DOI: 10.1016/j.jss.2022.111549.

Frattini, Julian; Fischbach, Jannik; Bauer, Andreas (2023): CiRA: An Open-Source Python Package for Automated Generation of Test Case Descriptions from Natural Language Requirements. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). Hannover, Germany, 04.09.2023 - 05.09.2023: IEEE, S. 68–71.

Gärtner, Alexander Elenga; Göhlich, Dietmar; Fay, Tu-Anh (2023): AUTOMATED CONDITION DETECTION IN REQUIREMENTS ENGINEERING. In: *Proc. Des. Soc.* 3, S. 707–716. DOI: 10.1017/pds.2023.71.

Gerdt, Angelina (2023): Systematische Literaturanalyse zu bestehenden Vorgehens- und Referenzmodellen zur Zuverlässigkeitsprognose von Heizsystemen. Projektarbeit. Technische Universität Dortmund. IT in Produktion und Logistik.

Gioia, Dennis A.; Corley, Kevin G.; Hamilton, Aimee L. (2013): Seeking Qualitative Rigor in Inductive Research. In: *Organizational Research Methods* 16 (1), S. 15–31. DOI: 10.1177/1094428112452151.

Google (2022): T5. Hg. v. Google. Online verfügbar unter https://huggingface.co/docs/transformers/model_doc/t5, zuletzt geprüft am 23.09.2024.

Goth, Greg (2009): Agile Tool Market Growing with the Philosophy. In: *IEEE Softw.* 26 (2), S. 88–91. DOI: 10.1109/MS.2009.30.

Goutte, Cyril; Gaussier, Eric (2005): A Probabilistic Interpretation of Precision, Recall and F1-Score. In: DBLP (Hg.): Advances in Information Retrieval. 27th European Conference on IR Research. Santiago de Compostela, Spain, March 21-23. DBLP, S. 345–359.

Gräßler, I.; Ozcan, D.; Preuß, D. (2023): AI-based extraction of requirements from regulations for automotive engineering. In: DS 125: Proceedings of the 34th Symposium Design for X (DFX2023). The 34th Symposium Design for X, 15 September 2023: The Design Society, S. 163–172.

Grigoleit, Sonja (2019): Natural Language Processing. Hg. v. Fraunhofer INT.

Gruber, John (2012): Markdown Syntax. In: *Daring Fireball* (38).

Guo, Jin L. C.; Steghöfer, Jan-Philipp; Vogelsang, Andreas; Cleland-Huang, Jane (2024): Natural Language Processing for Requirements Traceability. Online verfügbar unter http://arxiv.org/pdf/2405.10845v1.

Hastie, Shane (2015): Gartner and Software Advice examine Agile Lifecycle Management Tools. Hg. v. InfoQ. Online verfügbar unter https://www.infoq.com/news/2015/02/agile-management-tools/, zuletzt geprüft am 06.07.2024.

Heo, Jueun; Kwon, Gibeom; Kwak, Changwon; Lee, Seonah (2024): A Comparison of Pretrained Models for Classifying Issue Reports. In: *IEEE Access* 12, S. 79568–79584. DOI: 10.1109/ACCESS.2024.3408688.

Hirschberg, Julia; Manning, Christopher (2015): Advances in natural language processing. In: *Science* (349), S. 261–266.

IBM (2024): What Is Application Lifecycle Management (ALM)? Online verfügbar unter https://www.ibm.com/topics/application-lifecycle-management, zuletzt geprüft am 05.07.2024.

Jafari, Sanaz Mohammad; Yildirim, Savas; Cevik, Mucahit; Basar, Ayse (2023): Anaphoric Ambiguity Resolution in Software Requirement Texts. In: 2023 IEEE International Conference on Big Data (BigData). 2023 IEEE International Conference on Big Data (BigData). Sorrento, Italy, 15.12.2023 - 18.12.2023: IEEE, S. 4722–4730.

Just, Julian (2024): Natural language processing for innovation search – Reviewing an emerging non-human innovation intermediary. In: *Technovation* 129, S. 102883. DOI: 10.1016/j.technovation.2023.102883.

Khurana, Diksha; Koli, Aditya; Khatter, Kiran; Singh, Sukhdev (2023): Natural language processing: state of the art, current trends and challenges. In: *Multimedia tools and applications* 82 (3), S. 3713–3744. DOI: 10.1007/s11042-022-13428-4.

Klespitz, József; Bíró, Miklós; Levente, Kovács (2016): Evaluation criteria for application life cycle management systems in practice. In: Anikó Szakál (Hg.): SACI 2016. 11th IEEE International Symposium on Applied Computational Intelligence and Informatics : May 12-14, 2016, Timişoara, Romania : proceedings. Piscataway, NJ: IEEE, S. 469–472.

LaTeX (2024): Introduction to LaTeX. Online verfügbar unter https://www.latex-project.org/about/, zuletzt geprüft am 29.09.2024.

LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015): Deep learning. In: *Nature* 521 (7553), S. 436–444. DOI: 10.1038/nature14539.

Leong, Iat Tou; Barbosa, Raul (2023): Translating Natural Language Requirements to Formal Specifications: A Study on GPT and Symbolic NLP. In: 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). Porto, Portugal, 27.06.2023 - 30.06.2023: IEEE, S. 259–262.

Lewis, Mike; Liu, Yinham; Goyal, Naman; Ghazvininejad, Marjan; Mohamed, Abdelrahman; Ves Stoyanov, Omar Levy; Zettlemoyer, Luke (2019): BART: Denoising Sequence-to-Sequence Pre training for Natural Language Generation, Translation, and Comprehension. Hg. v. FacebookAI. Meta.

Li, Youjia; Shi, Jianjun; Zhang, Zheng (2024): An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering. In: *IEEE Access* 12, S. 53074–53087. DOI: 10.1109/ACCESS.2024.3385682.

Liddy, Elizabeth D. (2001): Natural Language Processing. In: *Encyclopedia of Library and Information Science*.

Liu, Yinhan; Ott, Myle; Goyal, Naman; Du Jingfei; Joshi, Mandar; Chen, Danqi et al. (2019): RoBERTa: A Robustly Optimized BERT Pretraining Approach. Online verfügbar unter http://arxiv.org/pdf/1907.11692v1.

Luitel, Dipeeka; Hassani, Shabnam; Sabetzadeh, Mehrdad (2024): Improving requirements completeness: automated assistance through large language models. In: *Requirements Eng* 29 (1), S. 73–95. DOI: 10.1007/s00766-024-00416-3.

Manning, Christopher; Surdeanu, Mihai; Bauer, John; Finkel, Jenny; Bethard, Steven; McClosky, David (2014a): The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, S. 55–60.

Manning, Christopher; Surdeanu, Mihai; Bauer, John; Finkel, Jenny; Bethard, Steven; McClosky, David (2014b): The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, S. 55–60. Online verfügbar unter https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf, zuletzt geprüft am 23.09.2024.

Mathur, Alok; Pradhan, Shreyaan; Soni, Prasoon; Patel, Dhruvil; Regunathan, Rajeshkannan (2023): Automated Test Case Generation Using T5 and GPT-3. In: 2023 9th International

Conference on Advanced Computing and Communication Systems (ICACCS). 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS). Coimbatore, India, 17.03.2023 - 18.03.2023: IEEE, S. 1986–1992.

Meta (2024): Introducing Llama 3.1- Our most capable models to date. Meta. Online verfügbar unter https://ai.meta.com/blog/meta-llama-3-1/, zuletzt geprüft am 23.09.2024.

Microsoft (2024): Überblick über die Application Lifecycle Management mit Microsoft Power Platform. Online verfügbar unter https://learn.microsoft.com/de-de/power-platform/alm/overview-alm, zuletzt geprüft am 05.07.2024.

Min, Bonan; Ross, Hayley; Sulem, Elior; Veyseh, Amir Pouran Ben; Nguyen, Thien Huu; Sainz, Oscar et al. (2021): Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey. Online verfügbar unter http://arxiv.org/pdf/2111.01243v1.

Mistral AI (2024): Mistral 7B - Frontier AI in your hands. Online verfügbar unter https://mistral.ai/news/announcing-mistral-7b/, zuletzt geprüft am 23.09.2024.

Nadkarni, Prakash M.; Ohno-Machado, Lucila; Chapman, Wendy W. (2011): Natural language processing: an introduction. In: *Journal of the American Medical Informatics Association : JAMIA* 18 (5), S. 544–551. DOI: 10.1136/amiajnl-2011-000464.

Nasiri, Samia; Lahmer, Mohammed (2024): AI-driven methodology for refining and clustering Agile requirements. In: 2024 4th International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET). 2024 4th International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET). FEZ, Morocco, 16.05.2024 - 17.05.2024: IEEE, S. 1–7.

NLTK Project (2024): NLTK - Natural Language Toolkit. Online verfügbar unter https://www.nltk.org/, zuletzt geprüft am 23.09.2024.

Ogundipe, Damilola Oluwaseun; Babatunde, Sodiq Odetunde; Abaku, Emmanuel Adeyemi (2024): AI AND PRODUCT MANAGEMENT: A THEORETICAL OVERVIEW FROM IDEA TO MARKET. In: *Int. j. manag. entrep. res* 6 (3), S. 950–969. DOI: 10.51594/ijmer.v6i3.965.

OpenAI (2024a): GPT-4o mini_ advancing cost-efficient intelligence. Online verfügbar unter https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, zuletzt geprüft am 23.09.2024.

OpenAI (2024b): Hello GPT-4o. Online verfügbar unter https://openai.com/index/hello-gpt-4o/, zuletzt geprüft am 23.09.2024.

Oswal, Jay U.; Kanakia, Harshil T.; Suktel, Devvrat (2024): Transforming Software Requirements into User Stories with GPT-3.5 -: An AI-Powered Approach. In: 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT). 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT). Bengaluru, India, 04.01.2024 - 06.01.2024: IEEE, S. 913–920.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B. and Grisel, O.; Blondel, M. et al. (2011): Scikit-learn: Machine learning in Python. In: *Journal of Machine Learning Research*, S. 2825–2830. Online verfügbar unter https://scikit-learn.org/stable/about.html#citing-scikit-learn, zuletzt geprüft am 23.09.2024.

Pichai, Sundar; Hassabis, Demis (2024): Introducing Gemini 1.5, Google_s next-generation AI model. Hg. v. Google. Online verfügbar unter https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/, zuletzt geprüft am 23.09.2024.

PTC (2024): Was ist Application Lifecycle Management (ALM. Online verfügbar unter https://www.ptc.com/de/technologies/application-lifecycle-management#:~:text=Die%20Anwendungslebenszyklus%2DVerwaltung%20(Application%20Lifecycle,und%20das%20Ende%20des%20Lebenszyklus., zuletzt geprüft am 05.07.2024.

PyTorch (2024): PyTorch 2.4. Online verfügbar unter https://pytorch.org/blog/pytorch2-4/, zuletzt geprüft am 23.09.2024.

Rahman, Kiramat; Ghani, Anwar; Alzahrani, Abdulrahman; Tariq, Muhammad Usman; Rahman, Arif Ur (2023): Pre-Trained Model-Based NFR Classification: Overcoming Limited Data Challenges. In: *IEEE Access* 11, S. 81787–81802. DOI: 10.1109/ACCESS.2023.3301725.

Regan, Gilbert; Biro, Miklos; Flood, Derek; McCaffery, Fergal (2015): Assessing Traceability – A Practical Experience and Lesson Learned. In: *Journal of Software: Evolution and Process 27.8*, S. 591–601.

Reimers, Nils; Gurevych, Iryna (2019): Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Online verfügbar unter http://arxiv.org/pdf/1908.10084v1.

Ronanki, Krishna; Berger, Christian; Horkoff, Jennifer (2023): Investigating ChatGPT's Potential to Assist in Requirements Elicitation Processes. In: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Durres, Albania, 06.09.2023 - 08.09.2023: IEEE, S. 354–361.

Ruan, Kun; Chen, Xiaohong; Jin, Zhi (2023): Requirements Modeling Aided by ChatGPT: An Experience in Embedded Systems. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). Hannover, Germany, 04.09.2023 - 05.09.2023: IEEE, S. 170–177.

Salesforce (2024): How Lifecycle Management Works. Online verfügbar unter https://help.salesforce.com/s/articleView?id=sf.lifecycle_mgmt_objects_and_process.htm&type=5, zuletzt geprüft am 05.07.2024.

Sanh, Victor; Debut, Lysandre; Chaumond, Julien; Wolf, Thomas (2019): DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. Online verfügbar unter http://arxiv.org/pdf/1910.01108v4.

SAP (2024): Application Lifecycle Management (ALM). Online verfügbar unter https://support.sap.com/en/alm.html, zuletzt geprüft am 05.07.2024.

Sawada, Kae; Pomerantz, Marc; Razo, Gus; Clark, Michael W. (2023): Intelligent requirement-to-test-case traceability system via Natural Language Processing and Machine Learning. In: 2023 IEEE 9th International Conference on Space Mission Challenges for Information Technology (SMC-IT). 2023 IEEE 9th International Conference on Space Mission Challenges for Information Technology (SMC-IT). Pasadena, CA, USA, 18.07.2023 - 27.07.2023: IEEE, S. 78–83.

Schramowski, Patrick; Turan, Cigdem; Andersen, Nico; Rothkopf, Constantin A.; Kersting, Kristian (2021): Large Pre-trained Language Models Contain Human-like Biases of What is Right and Wrong to Do. Online verfügbar unter http://arxiv.org/pdf/2103.11790v3.

Scoggin, S. B.; Torres Marques-Neto, H. (2024): Identifying Valid User Stories Using BERT Pre-trained Natural Language Models. In: Alvaro Rocha, Hojjat Adeli, Gintautas Dzemyda, Fernando Moreira und Valentina Colla (Hg.): Information Systems and Technologies. Cham: Springer Nature Switzerland (801), S. 167–177.

Scopus (2024): Scopus Search Guide. Online verfügbar unter https://schema.elsevier.com/dtds/document/bkapi/search/SCOPUSSearchTips.htm, zuletzt geprüft am 22.07.2024.

Seidel, P.; Späthe, S. (2024): Development and Validation of AI-Driven NLP Algorithms for Chatbots in Requirement Engineering. In: Frank Phillipson, Gerald Eichler, Christian Erfurth und Günter Fahrnberger (Hg.): Innovations for Community Services. Cham: Springer Nature Switzerland (2109), S. 132–149.

Siemens (2024): Application Lifecycle Management (ALM). Online verfügbar unter https://plm.sw.siemens.com/de-DE/polarion/application-lifecycle-management-alm/, zuletzt geprüft am 05.07.2024.

Simone, Vincenzo de; Amalfitano, Domenico; Fasolino, Anna Rita (2018): Exploiting ALM and MDE for Supporting Questionnaire-Based Gap Analysis Processes. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Prague, 29.08.2018 - 31.08.2018: IEEE, S. 1–8.

Sinzig, Corinna (2017): Nichtmarktstrategien multinationaler Unternehmen. Wiesbaden: Springer Fachmedien Wiesbaden.

TensorFlow (2024): TensorFlow. Hg. v. TensorFlow. Online verfügbar unter https://www.tensorflow.org/, zuletzt geprüft am 23.09.2024.

Uygun, Yilmaz; Momodu, Victor (2024): Local large language models to simplify requirement engineering documents in the automotive industry. In: *Production & Manufacturing Research* 12 (1), Artikel 2375296. DOI: 10.1080/21693277.2024.2375296.

Vito, Gabriele de; Palomba, Fabio; Gravino, Carmine; Di Martino, Sergio; Ferrucci, Filomena (2023): ECHO: An Approach to Enhance Use Case Quality Exploiting Large Language Models. In: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Durres, Albania, 06.09.2023 - 08.09.2023: IEEE, S. 53–60.

vom Brocke, Jan; Simons, Alexander; Niehaves, Bjoern; Reimer, Kai; Plattfaut, Ralf; Cleven, Anne (2009): Reconstructing the giant: on the importance of rigour in documenting the literature search process. In: *17th European Conference on Information Systems (ECIS 2009)*, S. 2206–2217. Online verfügbar unter http://aisel.aisnet.org/ecis2009/161/.

W3Schools (2024a): Introduction to HTML. Online verfügbar unter https://www.w3schools.com/html/html_intro.asp, zuletzt geprüft am 29.09.2024.

W3Schools (2024b): JSON Introduction. Online verfügbar unter https://www.w3schools.com/js/js_json_intro.asp, zuletzt geprüft am 29.09.2024.

W3Schools (2024c): XML Introduction. Online verfügbar unter https://www.w3schools.com/xml/xml_whatis.asp, zuletzt geprüft am 29.09.2024.

Wang, K.; Zhang, F.; Sabetzadeh, M. (2024): Automated Requirements Demarcation using Large Language Models: An Empirical Study. Online verfügbar unter https://www.scopus.com/inward/record.uri?eid=2-s2.0-85193072295&partnerID=40&md5=8d893066044617261a3e483b5c209c37.

Wang, Lei; Liu, Zhengchao; Liu, Ang; Tao, Fei (2021): Artificial intelligence in product lifecycle management. In: *Int J Adv Manuf Technol* 114 (3-4), S. 771–796. DOI: 10.1007/s00170-021-06882-1.

Wei, Jiahao; Chen, Xuan; Xiao, Hongbin; Tang, Shangzhi; Xie, Xiaolan; Li, Zhi (2023): Natural Language Processing-Based Requirements Modeling: A Case Study on Problem Frames. In: 2023 30th Asia-Pacific Software Engineering Conference (APSEC). 2023 30th Asia-Pacific Software Engineering Conference (APSEC). Seoul, Korea, Republic of, 04.12.2023 - 07.12.2023: IEEE, S. 191–200.

World Wide Web Consortium (1999): Web content accessibility guidelines 1.0. Online verfügbar unter http://www.w3.org/TR/WAI-WEBCONTENT/.

World Wide Web Consortium (2006): Namespaces in XML 1.1. Online verfügbar unter http://www.w3.org/TR/2006/REC-xml-names11-20060816/.

World Wide Web Consortium (2010): Xml path language (xpath) 2.0.

Wyrwich, Fabian; Kharatyan, Aschot; Dumitrescu, Roman (2024): Interdisciplinary system lifecycle management – a systematic literature review. In: *Proc. Des. Soc.* 4, S. 2765–2774. DOI: 10.1017/pds.2024.280.

Yeow, Jie Sh'ng; Rana, Muhammad Ehsan; Abdul Majid, Nur Amira (2024): An Automated Model of Software Requirement Engineering Using GPT-3.5. In: 2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS). 2024 ASU International Conference in Emerging Technologies for Sustainability and Intelligent Systems (ICETSIS). Manama, Bahrain, 28.01.2024 - 29.01.2024: IEEE, S. 1746–1755.

Young, Tom; Hazarika, Devamanyu; Poria, Soujanya; Cambria, Erik (2018): Recent Trends in Deep Learning Based Natural Language Processing [Review Article]. In: *IEEE Comput. Intell. Mag.* 13 (3), S. 55–75. DOI: 10.1109/MCI.2018.2840738.

Yuill, Simon; Halpin, Harry (2006): Python.

Zhang, Jianzhang; Chen, Yiyang; Liu, Chuang; Niu, Nan; Wang, Yinglin (2023a): Empirical Evaluation of ChatGPT on Requirements Information Retrieval Under Zero-Shot Setting. In: 2023 International Conference on Intelligent Computing and Next Generation Networks（ICNGN). 2023 International Conference on Intelligent Computing and Next Generation Networks（ICNGN). Hangzhou, China, 17.11.2023 - 18.11.2023: IEEE, S. 1–6.

Zhang, Li; Ouyang, Liubo; Qin, Jiahao (2023b): Automatic Detection Method for Software Requirements Text with Language Processing Model. In: 2023 6th International Conference on Software Engineering and Computer Science (CSECS). 2023 6th International Conference on Software Engineering and Computer Science (CSECS). Chengdu, China, 22.12.2023 - 24.12.2023: IEEE, S. 1–5.

Zhao, Liping; Alhoshan, Waad; Ferrari, Alessio; Letsholo, Keletso J.; Ajagbe, Muideen A.; Chioasca, Erol-Valeriu; Batista-Navarro, Riza T. (2022): Natural Language Processing for Requirements Engineering. In: *ACM Comput. Surv.* 54 (3), S. 1–41. DOI: 10.1145/3444689.

Anhang

Anhang A: Ergebnisse der systematischen Literaturrecherche

Tabelle A-6-1 Ergebnisse der Systematischen Literaturrecherche mit Hyperlinks

Database	Authors	Title	Year	Publication Type	Research Type	Field	ALM Field	Use Case	NLP Tool/Library/ Method	NLP Application	NLP Application2
Scopus	Alhoshan W., Ferrari A., Zhao L.	Zero-shot learning for requirements classification: An exploratory study	2023	Conference Paper	Exploratory Study	Software Engineering	Requirements Engineering	Requirements Classification using Zero-shot Learning	Sentence-BERT (Sbert), All-MiniLM-L12 (AllMini), Bert4RE, BERTOverflow (SObert)	Text Classification	
IEEExplore	Arulmohan, S., Meurs, M. J., & Mosser, S.	Extracting Domain Models from Textual Requirements in the Era of Large Language Models	2023	Conference Paper	Experimental Study	Software Engineering	Requirements Engineering	Extraction of Domain Models from Textual Requirements	GPT 3.5	Information Extraction	
Scopus; IEEExplore	Bertram V., Kausch H., Kusmenko E., Nqiri H., Rumpe B., Venhoff C.	Leveraging Natural Language Processing for a Consistency Checking Toolchain of Automotive Requirements	2023	Journal Article	Applied Research	Automotive Industry	Requirements Engineering	Consistency Checking of Automotive Requirements	GPT-J-6B, ML Algorithm	Information Extraction	Text Classification
IEEExplore	Boukhlif, M., Hanine, M., Kharmoum, N., Noriega, A. R., Obeso, D. G., & Ashraf, I.	Natural Language Processing-Based Software Testing: A Systematic Literature Review	2024	Journal Article	Literature Review	Software Testing	Test Management	Overview of NLP in Software Testing, including test case generation	BERT, Word2Vec, Stanford CoreNLP	Text Classification	Information Extraction
IEEExplore	Couder, J. O., Gomez, D., & Ochoa, O.	Requirements Verification Through the Analysis of Source Code by Large Language Models	2024	Conference Paper	Applied Research	Software Engineering	Requirements Engineering	Verification of requirements through analysis of source code using LLMs	GPT 3.5	Information Extraction	Text Classification
Scopus	Cruciani F., Moore S., Nugent C.	Comparing general purpose pre-trained Word and Sentence embeddings for Requirements Classification	2023	Workshop Paper	Experimental Study	Requirements Engineering	Requirements Engineering	Classification of functional and non-functional requirements	BERT, DistilBERT, SBERT, Universal Sentence Encoder, GloVe	Text Classification	
Scopus; IEEExplore	Das S., Deb N., Cortesi A., Chaki N.	Zero-shot Learning for Named Entity Recognition in Software Specification Documents	2023	Conference Paper	Experimental Study	Software Engineering	Document Management	Named Entity Recognition (NER) in software specifications	GPT-3, T5	Information Extraction	
IEEExplore	De Vito, G., Palomba, F., Gravino, C., Di Martino, S., & Ferrucci, F.	ECHO: An Approach to Enhance Use Case Quality Exploiting Large Language Models	2023	Conference Paper	Applied Research	Requirements Engineering	Requirements Engineering	Enhancing the quality of use cases	ChatGPT	Natural Language Generation	Information Extraction
IEEExplore	Dos Santos, K. Bouchard, F. Petrillo	AI-Driven User Story Generation	2024	Conference Paper	Applied Research	Agile Software Development	Requirements Engineering	Automatic generation of user stories in agile development	GPT-3, N-gram model	Natural Language Generation	
Scopus	El-Hajjami A., Fafin N., Salinesi C.	Which AI Technique Is Better to Classify Requirements? An	2024	Conference Paper	Experimental Study	Requirements Engineering	Requirements Engineering	Classification of functional and non-functional requirements	SVM, LSTM, GPT-3.5, GPT-4	Text Classification	

		Experiment with SVM, LSTM, and ChatGPT									
Scopus; IEEExplore; WebofScience	Ezzini S., Abualhaija S., Arora C., Sabetzadeh M.	AI-based Question Answering Assistance for Analyzing Natural-language Requirements	2023	Conference Paper	Applied Research	Requirements Engineering	Requirements Engineering	Providing question-answering assistance for analyzing natural language requirements	GPT-3	Natural Language Generation	Information Extraction
Scopus	Fischbach J., Frattini J., Vogelsang A., Mendez D., Unterkalmsteiner M., Wehrle A., Henao P.R., Yousefi P., Juricic T., Radduenz J., Wiecher C.	Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study	2023	Journal Article	Case Study	Software Testing	Test Management	Automatic generation of acceptance tests from requirements	BERT, RoBERTa	Information Extraction	
Scopus; IEEExplore	Frattini J., Fischbach J., Bauer A.	CiRA: An Open-Source Python Package for Automated Generation of Test Case Descriptions from Natural Language Requirements	2023	Conference Paper	Applied Research	Software Testing	Test Management	Automated generation of test case descriptions from natural language requirements	BERT, RoBERTa	Information Extraction	Text Classification
Scopus	Gärtner A.E., Göhlich D., Fay T.-A.	AUTOMATED CONDITION DETECTION IN REQUIREMENTS ENGINEERING	2023	Conference Paper	Applied Research	Requirements Engineering	Requirements Engineering	Detection of conditions within requirements	GPT-3, GPT-3.5, GPT-4, LLaMA	Information Extraction	Text Classification
Scopus	Gräßler I., Özcan D., Preuß D.	AI-based extraction of requirements from regulations for automotive engineering	2023	Conference Paper	Applied Research	Automotive Engineering	Requirements Engineering	Extraction of requirements from regulatory documents in automotive engineering	BERT-based model	Information Extraction	Text Classification
IEEExplore	Heo, J., Kwon, G., Kwak, C., & Lee, S.	A Comparison of Pretrained Models for Classifying Issue Reports	2024	Conference Paper	Experimental Study	Bug and Issue Management	Bug and Issue Management	Classification of issue reports using pretrained models	BERT, RoBERTa, GPT-3	Text Classification	
IEEExplore	Jafari, S. Yildirim, M. Cevik, A. Basar	Anaphoric Ambiguity Resolution in Software Requirement Texts	2023	Conference Paper	Applied Research	Requirements Engineering	Requirements Engineering	Resolving anaphoric ambiguity in software requirements texts	BERT, RoBERTa, GPT-3.5, GPT-4, T5	Information Extraction	
IEEExplore	Leong, I. T., & Barbosa, R.	Translating Natural Language Requirements to Formal Specifications: A Study on GPT and Symbolic NLP	2023	Conference Paper	Applied Research	Requirements Engineering	Requirements Engineering	Translating natural language requirements into formal specifications using GPT and symbolic methods	GPT (ChatGPT), ccg2lambda	Information Extraction	Natural Language Generation
IEEExplore	Li, J. Shi, Z. Zhang	An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering	2024	Conference Paper	Applied Research	Software Development	Requirements Engineering	Leveraging ChatGPT for source code generation and refinement using prompt engineering techniques	ChatGPT	Natural Language Generation	
Scopus; Web of Science	Luitel D., Hassani S., Sabetzadeh M.	Improving requirements completeness: automated assistance through large language models	2024	Book Chapter	Applied Research	Requirements Engineering	Requirements Engineering	Improving the completeness of requirements through automated assistance using large language models	GPT	Natural Language Generation	Information Extraction

IEEExplore	Mathur, A., Pradhan, S., Soni, P., Patel, D., & Regunathan, R.	Automated Test Case Generation Using T5 and GPT-3	2023	Conference Paper	Applied Research	Software Testing	Test Management	Automated generation of test cases using large language models like T5 and GPT-3	T5, GPT-3	Natural Language Generation	
Scopus; IEEExplore	Nasiri S., Lahmer M.	AI-driven methodology for refining and clustering Agile requirements	2024	Conference Paper	Applied Research	Agile Software Development	Requirements Engineering	Refining and clustering Agile user stories to identify duplicates and improve backlog management	SBERT (Sentence-BERT), SpaCy	Text Classification	Information Extraction
Scopus; IEEExplore	Oswal J.U., Kanakia H.T., Suktel D.	Transforming Software Requirements into User Stories with GPT-3.5 -: An AI-Powered Approach	2024	Conference Paper	Applied Research	Agile Software Development	Requirements Engineering	Converting software requirements into user stories using GPT-3.5	GPT-3.5	Natural Language Generation	
IEEExplore	Rahman, K., Ghani, A., Alzahrani, A., Tariq, M. U., & Rahman, A. U.	Pre-Trained Model-Based NFR Classification: Overcoming Limited Data Challenges	2023	Conference Paper	Experimental Research	Requirements Engineering	Requirements Engineering	Classification of non-functional requirements (NFR) using pre-trained models to overcome limited data challenges	BERT, GPT-2	Text Classification	
Scopus; IEEExplore	Ronanki K., Berger C., Horkoff J.	Investigating ChatGPT's Potential to Assist in Requirements Elicitation Processes	2023	Conference Paper	Experimental Research	Requirements Engineering	Requirements Engineering	Exploring the potential of ChatGPT to assist in requirements elicitation by comparing its output with human experts	ChatGPT	Natural Language Generation	Information Extraction
IEEExplore	Ruan, K., Chen, X., & Jin, Z.	Requirements Modeling Aided by ChatGPT: An Experience in Embedded Systems	2023	Conference Paper	Applied Research	Embedded Systems	Requirements Engineering	Automated generation of requirements models for embedded systems using ChatGPT and predefined composition rules	ChatGPT	Information Extraction	Natural Language Generation
Scopus; IEEExplore	Sawada K., Pomerantz M., Razo G., Clark M.W.	Intelligent requirement-to-test-case traceability system via Natural Language Processing and Machine Learning	2023	Conference Paper	Applied Research	Software Engineering	Test Management	Automating the mapping of software requirements to test cases using NLP and machine learning techniques to improve software reliability and reduce development time	BERT, SpaCy, NLTK, GPT-3 (local implementation)	Information Extraction	Text Classification
Scopus	Scoggin S.B., Torres Marques-Neto H.	Identifying Valid User Stories Using BERT Pre-trained Natural Language Models	2024	Conference Paper	Applied Research	Requirements Engineering	Requirements Engineering	Identification and validation of user stories	BERT	Text Classification	
Scopus	Seidel P., Späthe S.	Development and Validation of AI-Driven NLP Algorithms for Chatbots in Requirement Engineering	2024	Book Chapter	Applied Research	Requirements Engineering	Requirements Engineering	Development and validation of NLP algorithms for chatbots in requirements engineering	GPT-2, BERT	Information Extraction	Natural Language Generation
Scopus; Web of Science	Uygun Y., Momodu V.	Local large language models to simplify requirement engineering documents in the automotive industry	2024	Conference Paper	Applied Research	Automotive Requirements Engineering	Requirements Engineering	Simplification of complex requirement engineering documents using large language models in the automotive industry	Local GPT	Summarization	Information Extraction
Scopus	Wang K., Zhang F.,	Automated Requirements	2024	Conference Paper	Empirical Study (Applied	Requirements	Requirements	Automated demarcation of	DeBERTa, Llama 2, RoBERTa	Information	Text Classification

	Sabetzadeh M.	Demarcation using Large Language Models: An Empirical Study			Research)	Engineering	Engineering	requirements using large language models	and comibnation	Extraction	
Scopus; IEEExplore; WebofScience	Wei J., Chen X., Xiao H., Tang S., Xie X., Li Z.	Natural Language Processing-Based Requirements Modeling: A Case Study on Problem Frames	2023	Conference Paper	Case Study	Requirments Engineering	Requirments Engineering	Modeling requirements using NLP in the context of problem frames	BERT, ELMo	Information Extraction	
IEEExplore	Yeow, J. S. N., Rana, M. E., & Majid, N. A. A.	An Automated Model of Software Requirement Engineering Using GPT-3.5	2024	Conference Paper	Applied Research	Requirments Engineering	Requirments Engineering	Automating software requirement engineering tasks using GPT-3.5	GPT-3.5	Natural Languag e Generation	Information Extraction
IEEExplore	Zhang, L., Ouyang, L., & Qin, J.	Automatic Detection Method for Software Requirements Text with Language Processing Model	2023	Conference Paper	Applied Research	Requirments Engineering	Requirments Engineering	Detecting defects in software requirements texts, specifically identifying issues such as fuzzy, unverifiable, and incomplete requirements.	BERT, TextCNN	Text Classification	Information Extraction
Scopus; IEEExplore	Zhang J., Chen Y., Liu C., Niu N., Wang Y.	Empirical Evaluation of ChatGPT on Requirements Information Retrieval Under Zero-Shot Setting	2023	Conference Paper	Empirical Study (Experimental Research)	Requirments Engineering	Requirments Engineering	Evaluating the performance of ChatGPT in retrieving requirements information in a zero-shot setting	ChatGPT	Information Extraction	Summarization

Anhang B: Ergebnisse und Evaluation

Tabelle B-6-2 Beispielausgabe des zweiten Schritts des Test Case Generierungsprogramms

ID	[redacted]
Type	[redacted]
Title	[redacted]
Description	[redacted]
Combined_input	[redacted]
Generated_test_case	[redacted]

Tabelle B-6-3 Aufteilung der Anforderungen in ihre Komponenten

Requirement	Akteur	Aktion	Bedingung	Ziel	Technisches Detail	Abhängigkeiten	Edge Case
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]		[redacted]	[redacted]	[redacted]	[redacted]
	[redacted]	[redacted]	[redacted]	[redacted]		[redacted]	[redacted]
	[redacted]		[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

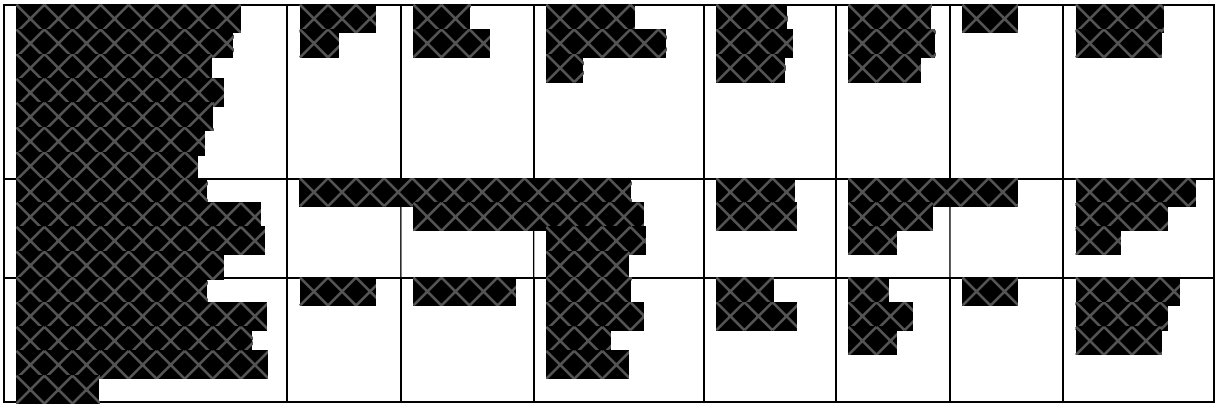
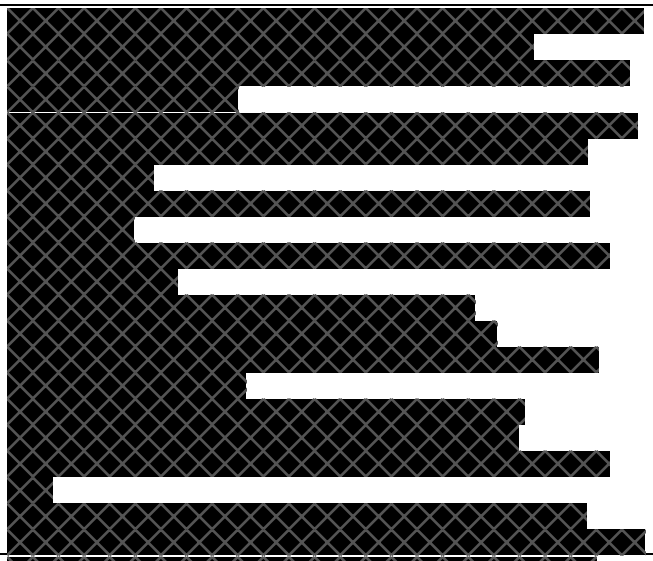
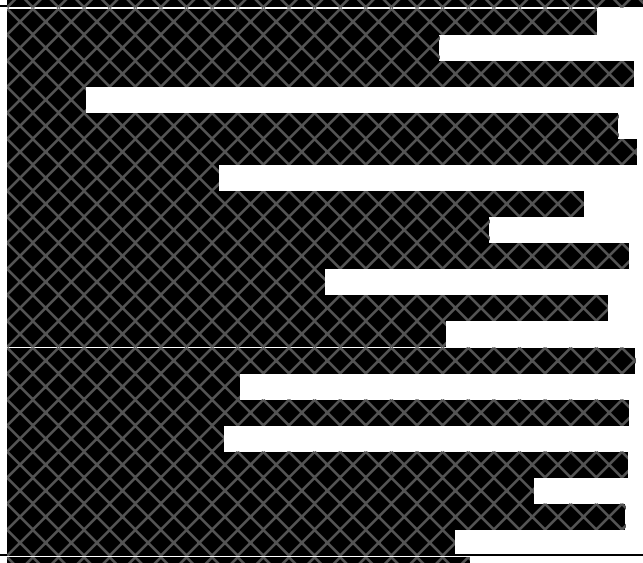
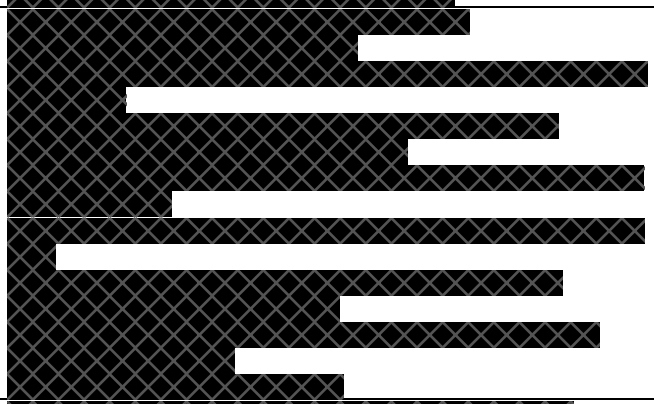
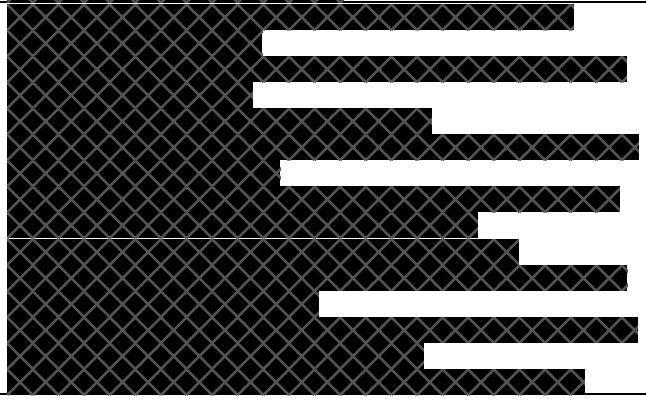


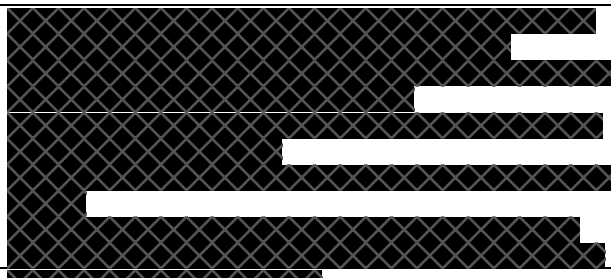
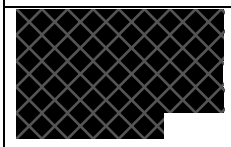
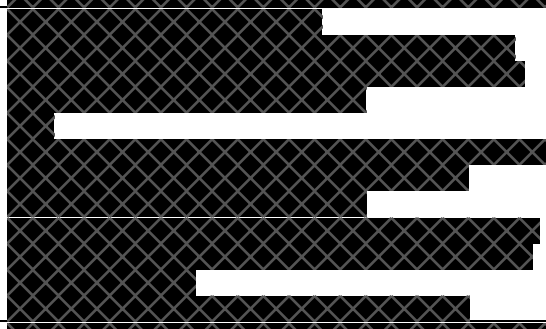
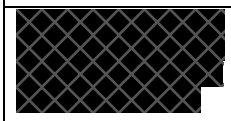
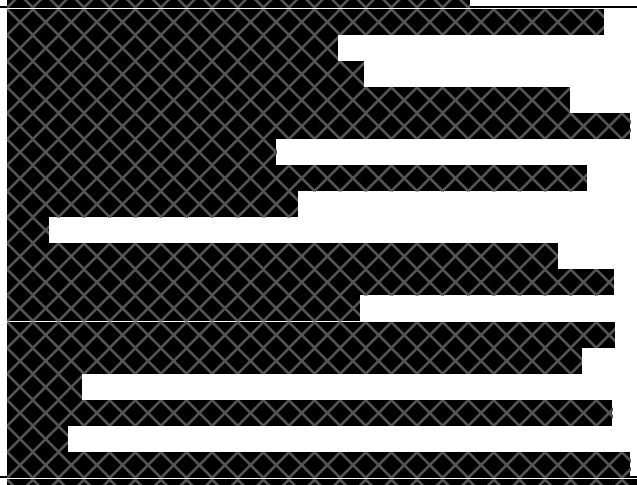
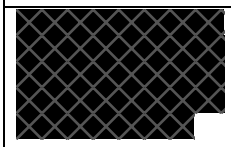
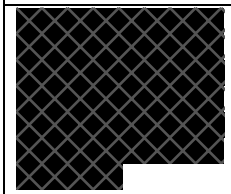
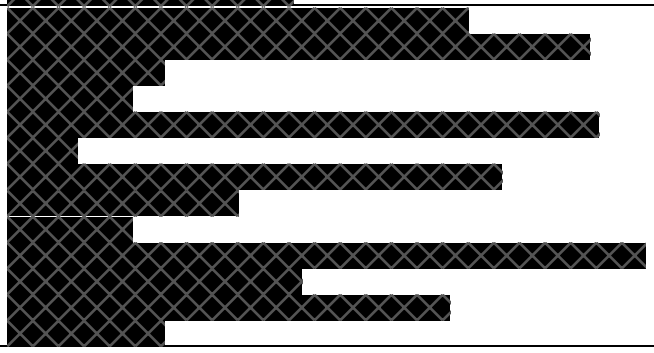
Tabelle B-6-4 Auswertung Precision, Recall und F1-Score der generierten Test Cases von GPT-4o

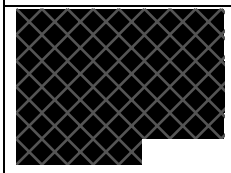
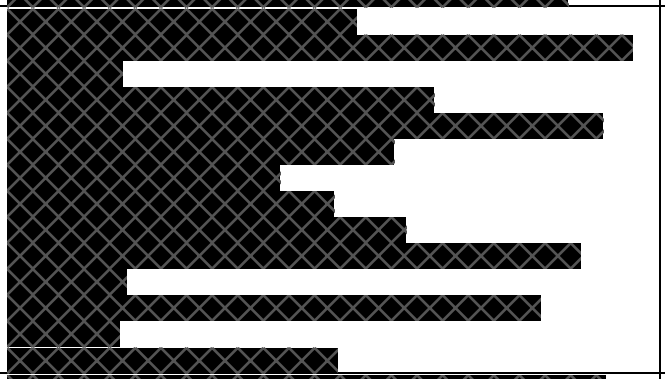
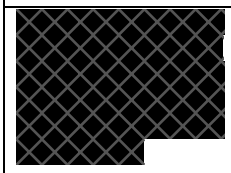
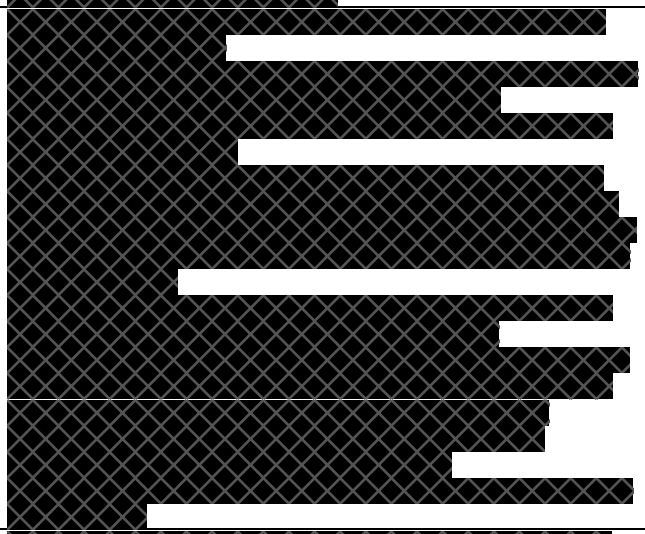
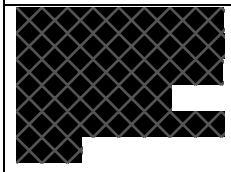
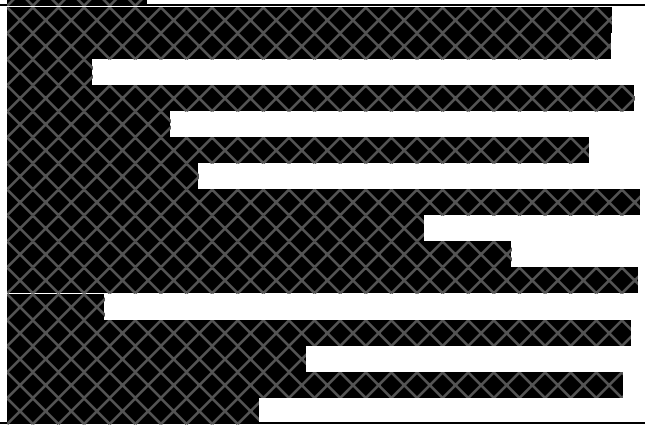
Requirement	Test Case	Precision	Recall	F1-Score
		1	1	1
		0,75	0,75	0,75

	[redacted]			
[redacted]	[redacted]	1	0,85714 2857	0,92307 6923
[redacted]	[redacted]	0,77777 7778	0,875	0,82352 9412
[redacted]	[redacted]	1	0,8	0,88888 8889

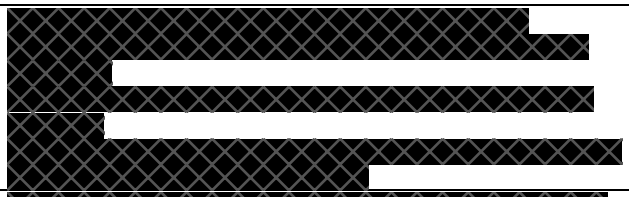
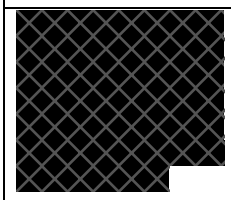
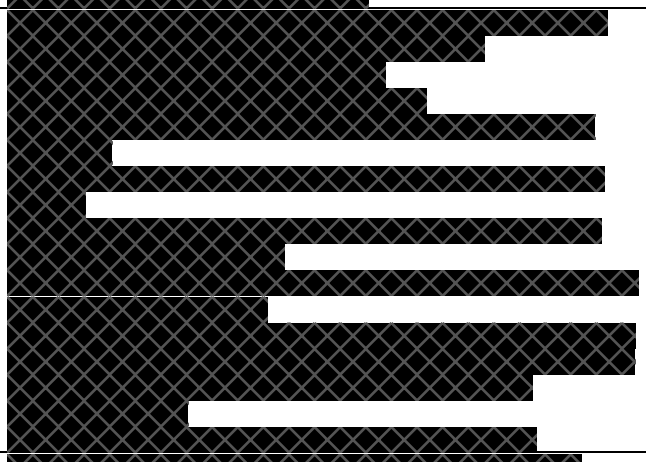
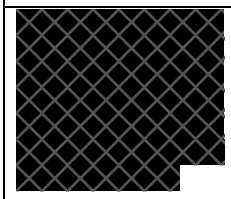
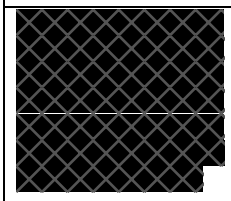
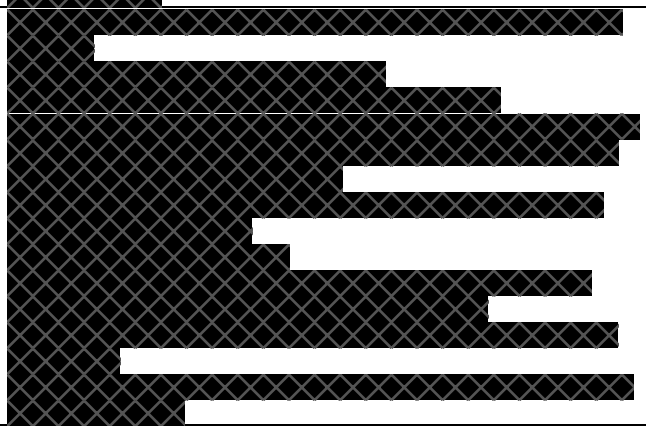
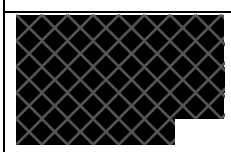
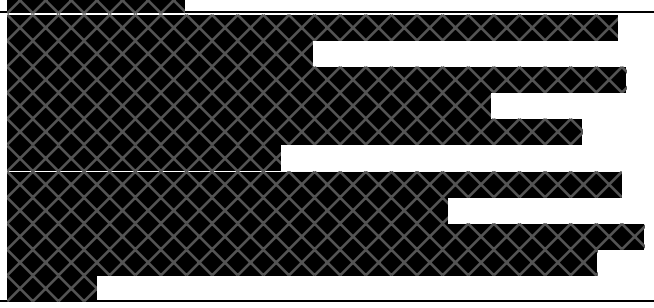
[redacted]	[redacted]	1	0,666666667	0,8
[redacted]	[redacted]	1	0,857142857	0,923076923
[redacted]	[redacted]	1	0,571428571	0,727272727
[redacted]	[redacted]	1	0,833333333	0,909090909

[redacted]		1	1	1
[redacted]		1	0,833333333	0,909090909
[redacted]		1	0,833333333	0,909090909
[redacted]		1	0,833333333	0,909090909

				
		1	0,833333333	0,909090909
		1	0,833333333	0,909090909
		1	1	1
		1	0,833333333	0,909090909

				
		1	0,8	0,888888889
		1	1	1
		1	1	1

		1	1	1
		1	1	1
		0,83333 3333	0,83333 3333	0,83333 3333

				
		1	1	1
		1	1	1
		1	1	1
		1	1	1

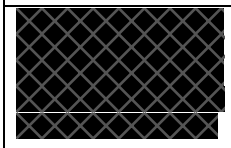
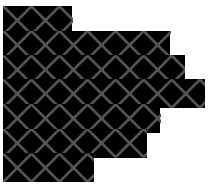
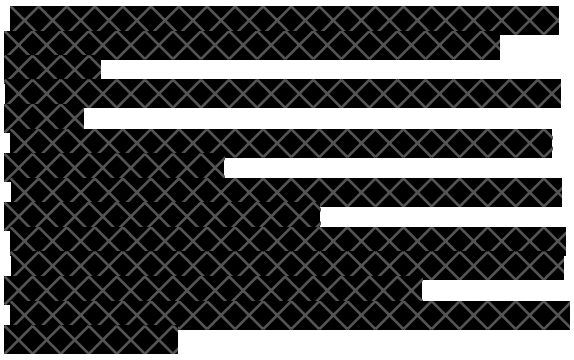
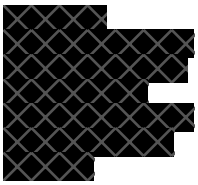
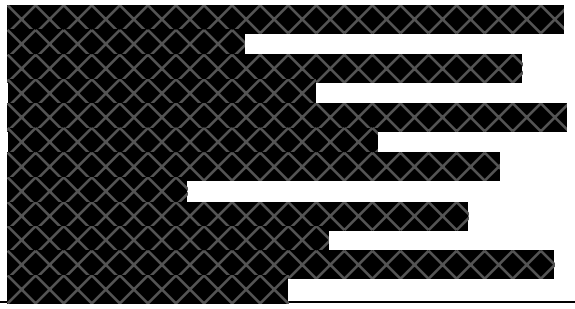
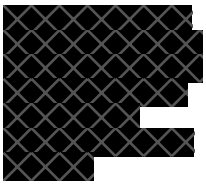
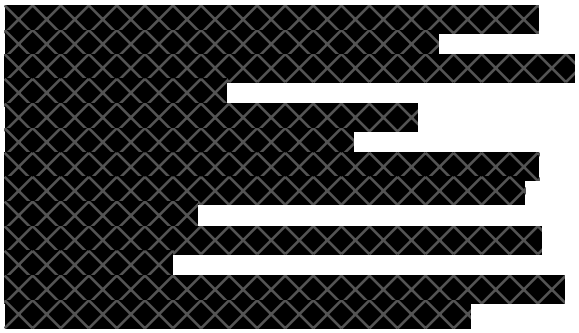
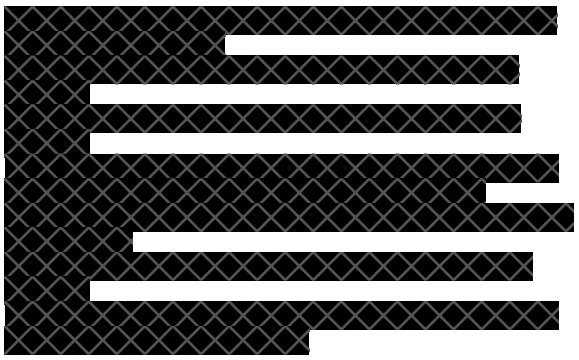
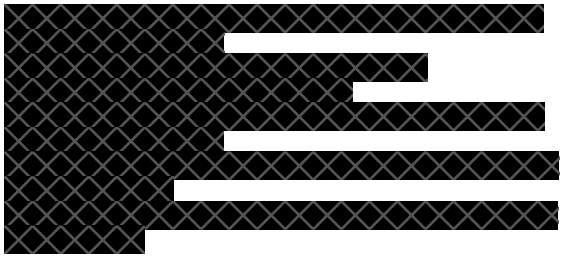
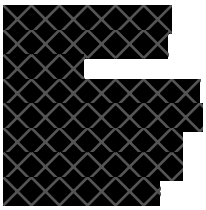
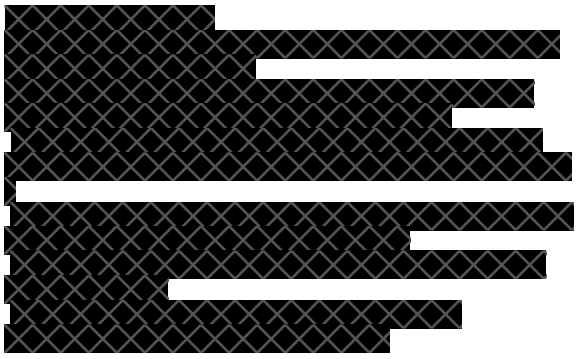
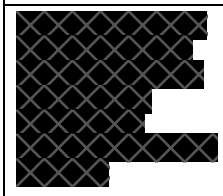
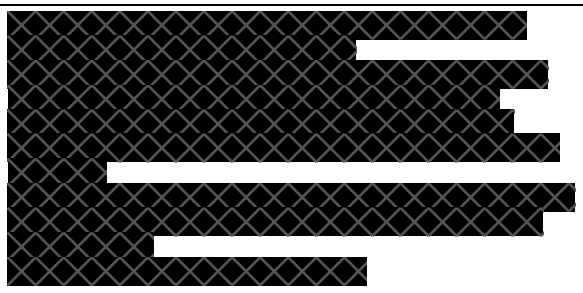
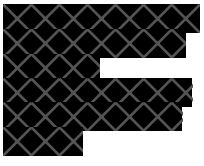
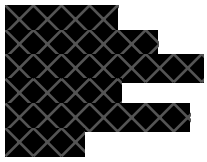
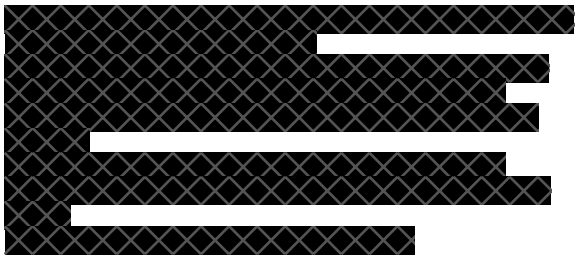
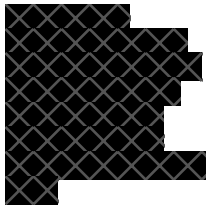
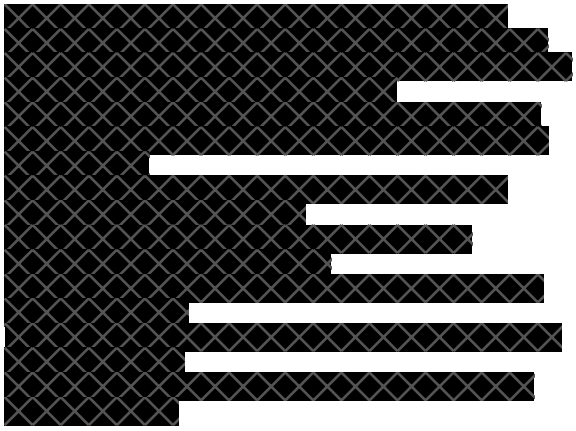
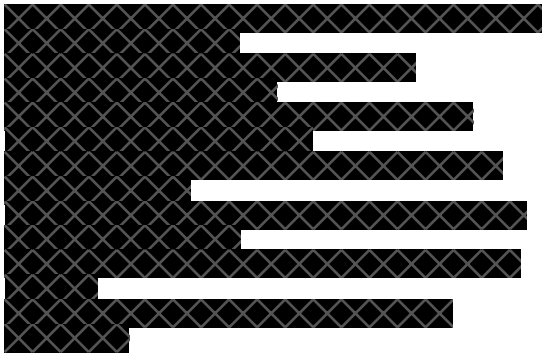
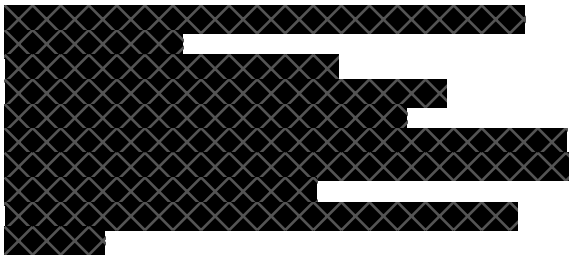
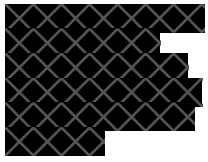
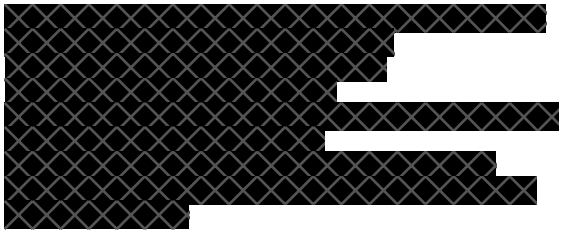
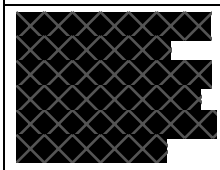
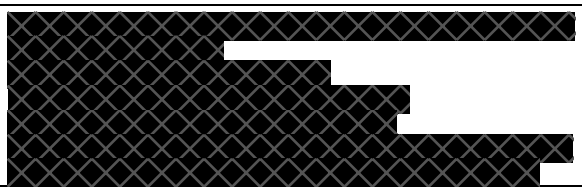
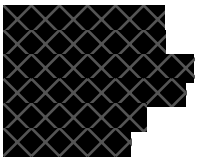
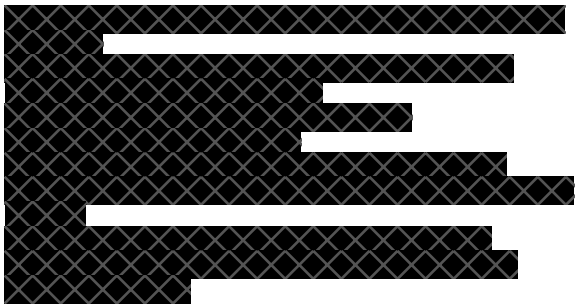
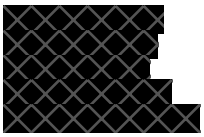
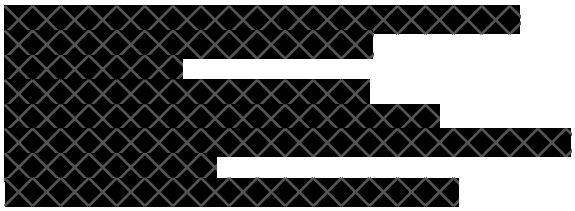
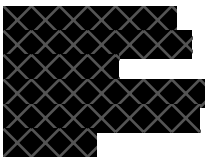
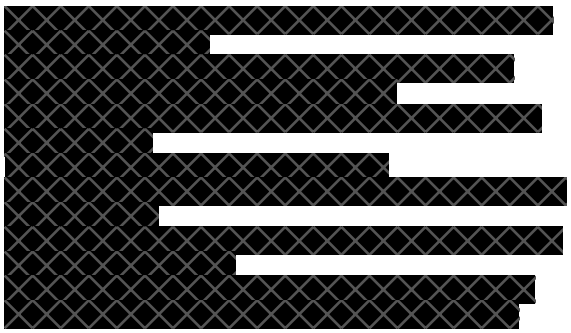
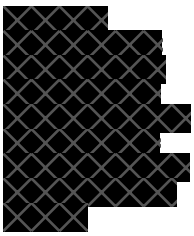
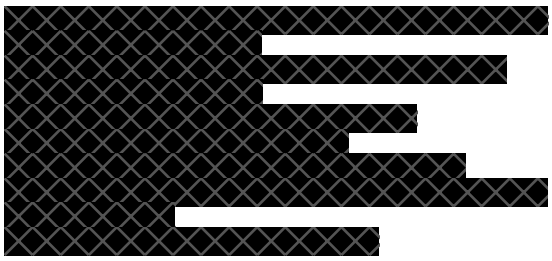
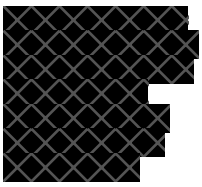
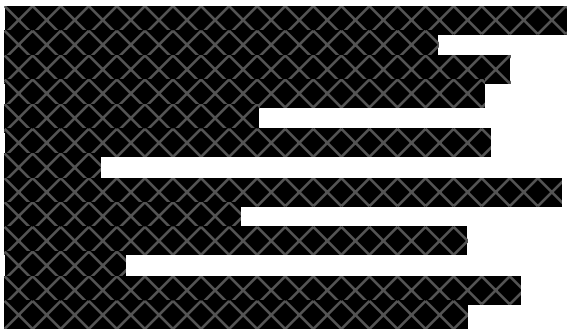
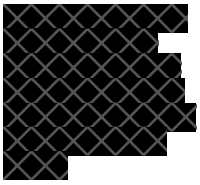
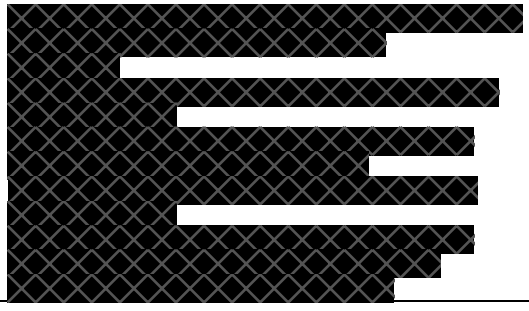
				
		1	1	1

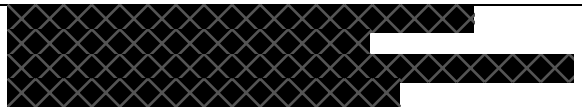
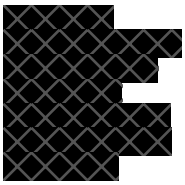
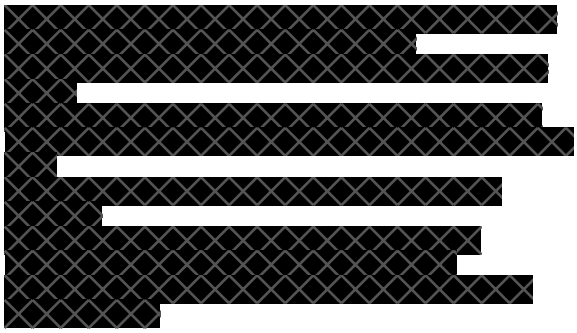
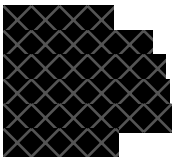
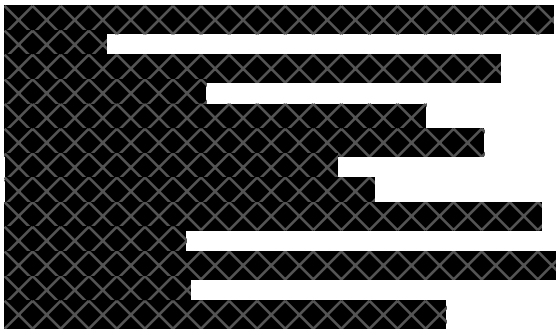
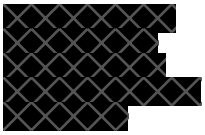
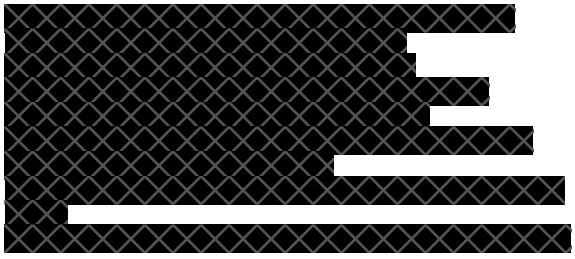
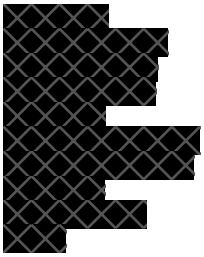
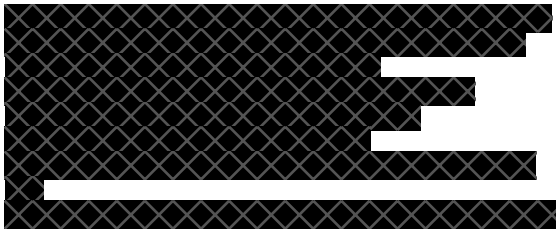
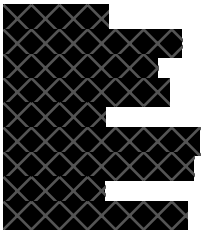
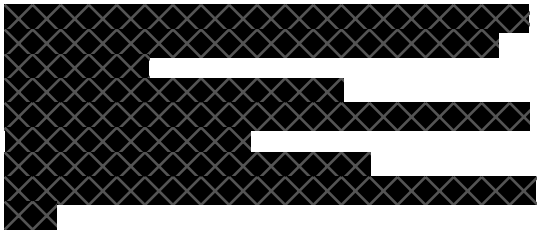
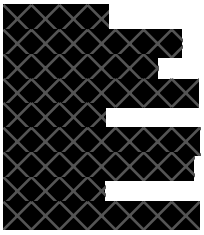
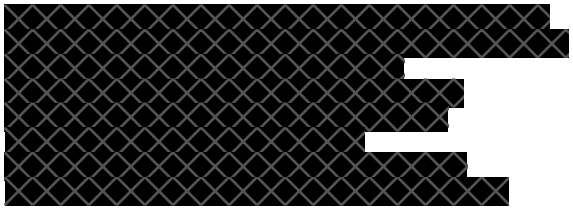
Tabelle B-6-5 Auswertung Precision, Recall und F1-Score der generierten Test Cases des spezialisierten ToolsI

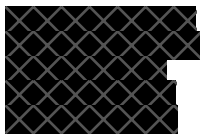
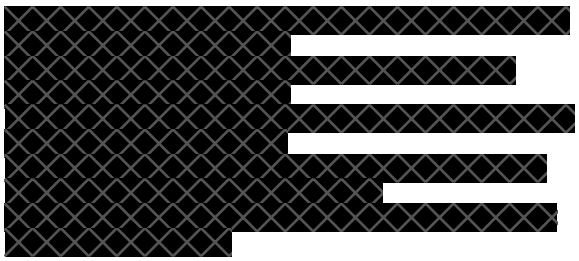
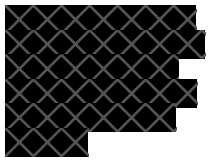
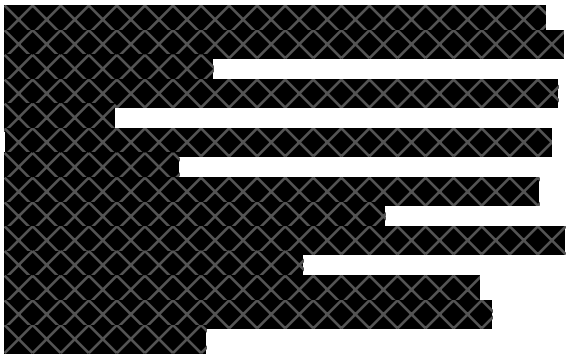
Requirement	Test Case	Precision	Recall	F1-Score
		1	1	1
		1	0,8	0,888889

		0,857143	0,857143	0,857143
		0,714286	0,714286	0,714286
		1	1	1
		0,714286	0,714286	0,714286
		0,714286	0,714286	0,714286
		0,571429	0,571429	0,571429

		1	1	1
		1	1	1
		0,875	1	0,933333
		1	1	1
		1	1	1
		1	1	1

		1	1	1
		1	1	1
		1	1	1
		0,857143	0,857143	0,857143
		0,857143	0,857143	0,857143
		0,666667	0,857143	0,75

				
		0,666667	0,666667	0,666667
		0,833333	0,833333	0,833333
		0,666667	0,666667	0,666667
		1	1	1
		1	1	1
		1	1	1

		0,666667	0,666667	0,666667
		0,5	0,666667	0,571429