h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Informatik –

AI-assisted Software Engineering: Potentialanalyse, Nutzungskonzept und Best Practices

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

vorgelegt von

Bennet Wilhelm

Matrikelnummer: 751205

Referent	:	Prof. Dr. Markus Voß
Korreferent	:	Prof. Dr. Bernhard Humm

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 16. Oktober 2023

Bennet Wilhelm

ABSTRACT

The advent of Large Language Models (LLMs) is a technical revolution with the potential to fundamentally change the field of software engineering. Tools based on these models, such as ChatGPT and GitHub Copilot, have unprecedented abilities that allow them to support and even automate knowledge work in many areas: summarizing texts, writing emails, or generating source code. However, the competencies and limitations in the practical application of these tools in software engineering have not yet been comprehensively investigated. A deeper understanding of how LLM-based tools can already support software engineering today, as well as an overview of the affected areas of application, would allow for a more targeted use of the tools and thus create added value for software engineering as a whole. As illustrated in this thesis, tools based on large language models can support software engineering in a productive way today already. The tools can be used in many use cases to increase the quality and quantity in software development. A more effective and efficient software engineering has far-reaching social and market-economic implications: from economic growth of people productivity to greater availability of software in all parts of the world to increased competitiveness of companies that use these tools in a targeted manner.

This work presents a wide-ranging analysis of the potential for the use of LLM-based tools in software engineering today. The identified potentials were tested in real project scenarios within an empirical pilot study. The results of the investigation are presented in the form of a usage concept, which presents concrete hints and best practices for the use of LLM-based tools in software engineering. Amongst the best practices are some definite recommendations, which have emerged especially positively in the pilot studies. Definite recommendations for the use of ChatGPT are the understanding of architecture patterns & styles, the creation of interface schemas, the coding with domain-specific languages, as well as the setup of the working environment. Definite recommendations for the use of GitHub Copilot are the commenting of source code and the creation of test data.

ZUSAMMENFASSUNG

Das Aufkommen großer Sprachmodelle (englisch: Large Language Models LLMs) ist eine technische Revolution mit dem Potenzial, das Software Engineering nachhaltig zu verändern. Auf diesen Modellen basierende Werkzeuge wie ChatGPT und GitHub Copilot verfügen über davor nie dagewesene Kompetenzen, welche es ihnen erlauben, die Wissensarbeit in vielen Bereichen zu unterstützen und teilweise sogar zu automatisieren: das Zusammenfassen von Texten, das Schreiben von E-Mails oder das Erstellen von Quellcode. Doch die Kompetenzen und Limitationen in der praktischen Anwendung dieser Werkzeuge im Software Engineering wurden bislang nicht umfassend untersucht. Ein tieferes Verständnis von der Art und Weise, wie LLM-basierte Werkzeuge das Software Engineering bereits heute unterstützen können, sowie ein Überblick über die betroffenen Aufgabenbereiche, würde einen gezielteren Einsatz der Werkzeuge erlauben und so einen Mehrwert für das Software Engineering als Ganzes schaffen. Bereits heute können auf großen Sprachmodellen basierende Werkzeuge das Software Engineering gewinnbringend unterstützen, wie diese Arbeit zeigt. Dabei können die Werkzeuge in vielen Anwendungsfällen eingesetzt werden, um die Qualität und die Quantität in der Softwareentwicklung zu steigern. Ein effektiveres und effizienteres Software Engineering hat umfassende gesellschaftliche und marktwirtschaftliche Implikationen: Von wirtschaftlichem Wachstum über höhere Verfügbarkeit von Software in allen Teilen der Welt bis hin zur gesteigerten Wettbewerbsfähigkeit von Unternehmen, welche diese Werkzeuge gezielt einsetzen.

Diese Arbeit stellt eine breitgefächerte Potenzialanalyse vor, welche das Potenzial für den Einsatz LLM-basierter Werkzeuge im Software Engineering heute evaluiert. Die identifizierten Potenziale wurden mittels einer empirischen Pilotuntersuchung in realen Projektszenarien erprobt. Die Ergebnisse der Untersuchung werden in Form eines Nutzungskonzepts vorgestellt, welches konkrete Hinweise und Best Practices für den Einsatz LLM-basierter Werkzeuge im Software Engineering darlegt. Unter den Best Practices sind einige klare Empfehlungen, welche in den Pilotuntersuchungen besonders positiv hervorgetreten sind. Klare Empfehlungen für den Einsatz von ChatGPT sind das Verstehen von Architekturpatterns & -Stile, das Erstellen von Schnittstellenschemata, das Coding mit Domain-Specific Languages, sowie die Einrichtung der Arbeitsumgebung. Klare Empfehlungen für den Einsatz von GitHub Copilot sind das Kommentieren von Quellcode und das Erstellen von Testdaten.

DANKSAGUNG

In erster Linie danke ich Accso für die Möglichkeit, in Kooperation mit ihnen am Thema des KI-gestützten Software Engineering zu forschen und für ihre Unterstützung, die mir während dieses Zeitraums zuteilwurde. Ich hoffe, die Erkenntnisse dieser Thesis kommen Accso zugute und finden auch außerhalb des Kreises der Probanden praktische Anwendung. Des Weiteren danke ich allen Probanden für ihre Teilnahme an den Pilotuntersuchungen, für ihre Zeit und für den spannenden Austausch in den Interviews. Besonderer Dank gilt Prof. Dr. Markus Voß (dem Referenten dieser Arbeit) und Dennis Grammes, die mich von inhaltlicher Seite, von fachlicher Seite und während des gesamten Prozesses unterstützt und betreut haben, und mir viele wertvolle Rückmeldungen gegeben haben. Ich danke Prof. Dr. Bernhard Humm (dem Coreferenten dieser Arbeit) für die interessanten Impulse und den regelmäßigen Austausch.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

API	Application Programming Interface
UML	Unified Modeling Language
LLM	Large Language Model
NLP	Natural Language Processing
KI	Künstliche Intelligenz
ML	Maschinelles Lernen
ANN	Artificial Neural Network
SQL	Structured Query Language
PC	Personal Computer
SE	Software Engineering
JSON	JavaScript Object Notation
SVG	Scalable Vector Graphics
IDE	Integrated Development Environment
IaC	Infrastructure as Code
CI/CD	Continuous Integration / Continuous Delivery
DSL	Domain-Specific Language
DNL	Deep Neural Learning
DNN	Deep Neural Network
LM	Language Modeling
SLM	Statistical Language Model
NLM	Neural Language Model
PLM	Pre-trained Language Model
CoT	Chain of Thought
RLHF	Reinforcement Learning Human Feedback
CLI	Command Line Interface
WEIRD	Western Educated Industrialized Rich Democratic

Teil I

THESIS

1 EINLEITUNG

1.1 MOTIVATION

Die Veröffentlichung von ChatGPT im November 2022 eröffnete eine neue Dimension der Kommunikationskompetenz von Künstliche Intelligenz (KI)-Systemen. Erstmals war ein Computersystem in der breiten Bevölkerung angekommen, welches Konversationen auf einem menschenähnlichen Niveau führen und komplexe Fragestellungen augenscheinlich durchdacht beantworten konnte. Der Chatbot basiert auf künstlicher Intelligenz (KI) und verfügt über ein breites Spektrum an Fähigkeiten. Er kann Fragen über seine Funktionsweise beantworten, Gedichte über beliebige Themen verfassen, komplizierte Texte in leichter Sprache zusammenfassen und sogar Computercode schreiben. Innerhalb von nur fünf Tagen registrierten sich eine Million Nutzer für die Benutzung des Chatbots. Im Vergleich dazu brauchten andere große Online-Dienste einige Monate oder gar Jahre, um diesen Meilenstein zu erreichen [Jan23], wie in Abbildung 1.1 illustriert. Auch in den Medien ist ChatGPT eine andauernd behandelte Thematik. So wurde der Chatbot in den ersten 6 Monaten nach dessen Veröffentlichung in 12 von 14 erschienenen Ausgaben der renommierten und deutschlandweit auflagenstärksten [Wik23b] Computerzeitschrift *c't* erwähnt[1]. In einem Interview mit Business Insider gab Bill Gates an, er betrachtet den KI-Chatbot als ebenso bedeutend wie die Einführung von Personal Computers (PCs) oder des Internets [Kay23].

Dreh- und Angelpunkt für die vielseitige Kommunikationskompetenz von ChatGPT ist, dass der Chatbot auf sogenannten großen Sprachmodellen basiert. Allgemein sind Sprachmodelle ein Ansatz auf dem Gebiet der künstlichen Intelligenz mit dem Ziel, Texte verschiedenster Art zu generieren. Über die letzten Jahre werden diese Sprachmodelle zunehmend größer und intelligenter, sodass LLMs — im Folgenden auch mit dem englischsprachigen Fachbegriff, Large Language Model (LLM), bezeichnet — ein zunehmend aktiveres Forschungsfeld darstellen [Zha+23, S. 3ff], welches äußerst rapide technische Fortschritte verzeichnet. Diese Fortschritte setzen völlig neue Maßstäbe für die sprachverarbeitenden Fähigkeiten von KI-Systemen. Und obwohl Sprachmodelle primär nur auf Text operieren, haben LLMs ein enormes Potenzial für weitaus abstraktere Herausforderungen. So stellen Wang et al. beispielsweise in [Wan+23] einen Agenten auf Basis des großen Sprachmodells *GPT-4* vor, der in dem Videospiel *Minecraft* eine komplexe Welt navigiert, sich eigenständig Ziele setzt und eigenständig Strategien zur Bewältigung verschiedenster Aufgaben erarbeitet.

Insbesondere die Softwareentwicklung stellt ein vielversprechendes Einsatzfeld für LLMs dar und erlebt einen regelrechten Hype. Werkzeuge wie

1 Siehe Abschnitt A.1 für eine detaillierte Aufführung der entsprechenden Ausgaben.

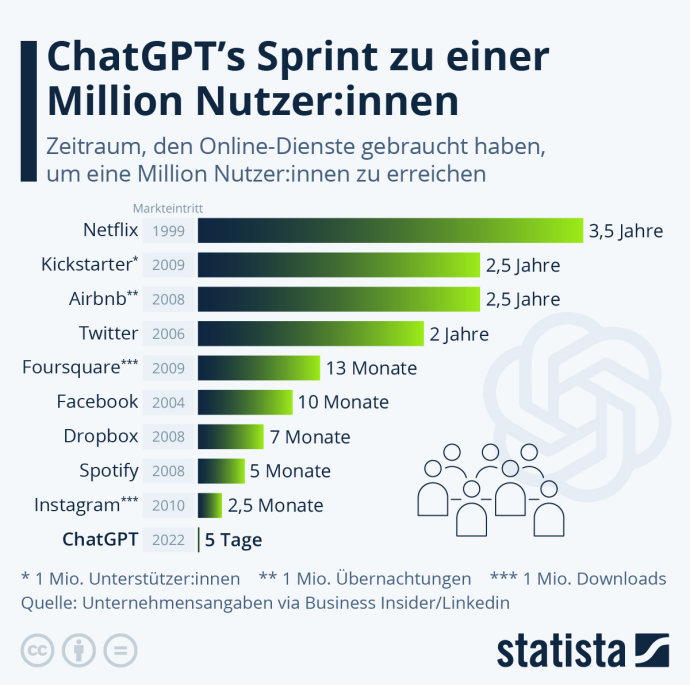


Abbildung 1.1: Diese Grafik stellt den Zeitraum gegenüber, welchen Onlinedienste zum Erreichen von einer Million Nutzer gebraucht haben. Während andere Dienste einige Monate oder gar Jahre benötigten, um diesen Meilenstein zu erreichen, knackte ihn ChatGPT innerhalb von nur 5 Tagen. Die Grafik stammt aus [Jan23].

GitHub Copilot und Amazon CodeWhisperer sind bereits heute in der Lage, allein aus der textuellen Beschreibung von Funktionalität, Quellcode zu generieren, der selbige Funktionalität implementiert. Der Chatbot ChatGPT kann gegebenen Quellcode auf Fehler prüfen und sogar aussagekräftige Kommentare für beliebigen Quellcode generieren. Viele der Tätigkeiten, welche ehemals nur von Menschen bewältigt werden konnten, scheinen bereits heute in Reichweite der Kompetenz von großen Sprachmodellen. Das Fazit steht noch aus, jedoch scheinen die rasanten und massiven Fortschritte der letzten Jahre eine Art Revolution in Gang gesetzt zu haben. Die Art wie wir arbeiten und wie wir Software entwickeln scheint im Begriff, sich nachhaltig zu verändern. Doch gehen mit den jüngsten, großen Fortschritten auf dem Gebiet der Sprachmodelle auch große Herausforderungen und signifikante Limitationen einher. Große Sprachmodelle sind eine Black Box, deren Funktionsweise nicht vollständig verstanden ist. Es ist unklar, wie die Modelle zu ihren Ergebnissen kommen und welcher Bias sich in diesem Prozess niederschlägt. Gerade aufgrund der hohen Kommunikationskompetenz von LLMs ist es für Menschen schwer, deren Limitationen nachzuvollziehen. Auch greifen LLM-basierte Werkzeuge tief in den Software Engineering Prozess ein, was Daten- und Geheimnisschutzbedenken aufwirft. So haben Amazon und Apple beispielsweise ihren Mitarbeitern die Entwicklung mit einigen KI-Werkzeugen aufgrund von Datenschutzbedenken zeitweise untersagt [Don23; Lin23]. In

diesem Kontext stellt diese Arbeit eine Untersuchung des Einsatzes von LLM-basierten Werkzeugen im Software Engineering (kurz: SE) vor.

1.2 ZIELE DER ARBEIT

Das Aufkommen großer Sprachmodelle ist eine technische Revolution mit dem Potenzial, die Art und Weise wie Software entwickelt wird, nachhaltig zu verändern. Die Unterstützung des Software Engineering durch LLM-basierte Werkzeuge wirft zuvor nie dagewesen neue Möglichkeiten auf, welche bis dato nicht umfassend untersucht wurden. Diese Möglichkeiten gewinnbringend einzusetzen ist eine signifikante Chance für das Software Engineering als Ganzes, für Unternehmen und für Personen, die an der Entwicklung von Software beteiligt sind. Eine Steigerung der Qualität oder der Quantität von entwickelter Software wäre von hoher marktwirtschaftlicher Relevanz und würde die Wettbewerbsfähigkeit der Unternehmen steigern, welche diese Werkzeuge einsetzen.

Das Ziel dieser Arbeit ist eine Bestandsaufnahme, inwiefern LLM-basierte Werkzeuge bereits heute gewinnbringend im Software Engineering eingesetzt werden können. Dazu präsentiert diese Arbeit eine fundierte Bewertung auf Basis empirischer Untersuchungen des Einsatzes der Werkzeuge in der Praxis. Die Ergebnisse der Untersuchungen werden in Form eines Nutzungskonzepts mit konkreten Hinweisen und Best Practices für den Einsatz der Werkzeuge vorgestellt.

Im Rahmen dieser Arbeit wird der gesamte Prozess des Software Engineering (SE) betrachtet. Es geht nicht nur um das Schreiben von Quellcode oder die Implementierung von Funktionalität. Der Anspruch ist, alle Aufgabenbereiche des Software Engineering zu betrachtet. Dazu zählen unter anderem Anforderungsanalyse, Softwarearchitektur, Testing und Projektmanagement. Ein gewinnbringender Einsatz dieser Werkzeuge bedeutet, dass ein signifikanter Mehrwert für das SE entsteht. Die genaue Natur dieses Mehrwerts ist keine triviale Frage und wird in Abschnitt 3.1 näher diskutiert.

Diese Arbeit richtet sich in erster Linie an alle Personen, die an der Entwicklung von Software beteiligt sind. Sie richtet sich an Softwareentwicklerinnen, an Projektleiter, an Studierende, an Stakeholder und an Interessierte. Sie soll einen verständlichen Einstieg in die Thematik bieten und gleichzeitig ein konkretes, praxisorientiertes Nutzungskonzept für die Adaption von KI-basierten Werkzeugen in den SE-Alltag darlegen.

1.3 METHODIK

Um die Ziele der Arbeit zu erreichen und die Potenziale des Einsatzes KI-basierter Werkzeuge im Software Engineering zu evaluieren wurde zunächst eine bereit gefächerte Potenzialanalyse durchgeführt. Dabei wurden zum einen konkrete Aufgabengebiete identifiziert, in welchen die Kompetenzen großer Sprachmodelle bereits heute das Software Engineering gewinnbringend unterstützen können. Zum anderen wurde ein breites Spektrum an heute ver-

fügbaren Werkzeugen auf ihre Eignung für den Einsatz im Softwareengineering sowie auf ihre Eignung für eine praktische Erprobung evaluiert. Diese Zwischenergebnisse dienen als Grundlage für praktische Pilotuntersuchungen im Umfeld professioneller, kommerzieller Softwareentwicklung. Hierbei erprobten 19 Software Engineers eine Auswahl der zuvor identifizierten Werkzeuge in realen Projektszenarien. Ihre Erfahrungen wurden mittels Experteninterviews erfasst und im Nachgang auf Basis verschiedener Kriterien ausgewertet. Letztendlich wurden die Ergebnisse der Pilotuntersuchungen zu einem Nutzungskonzept zusammengeführt, welche konkrete Hinweise und Best Practices für den Einsatz LLM-basierter Werkzeuge im Software Engineering darlegt.

1.4 GLIEDERUNG

Die Einleitung in die vorliegende Arbeit findet in Kapitel 1 statt. Dort sind die Motivation, die Ziele der Arbeit, sowie ein Überblick über Methodik und Gliederung aufgeführt. Kapitel 2 stellt die Grundlagen und theoretischen Hintergründe dar, welche für diese Arbeit relevant sind. Dabei wird auf die Begriffe der künstlichen Intelligenz (KI), der Verarbeitung natürlicher Sprache (NLP), des maschinellen Lernens (ML), sowie der großen Sprachmodelle (LLMs) eingegangen. Insbesondere wird auf die Kompetenzen und Limitationen großer Sprachmodelle eingegangen. Kapitel 3 präsentiert die Potenzialanalyse, welche als Grundlage für die Pilotuntersuchungen dient. Dort ist die verwendete Methodik beschrieben und es werden zwei Ansätze zur Identifikation von Potenzialen vorgestellt. Zuletzt werden dort die Ergebnisse zu einem Fokus für die weiteren Untersuchungen zusammengeführt. Die Pilotuntersuchungen selbst sind in Kapitel 4 dargestellt. Darin ist die Versuchsbeschreibung detailliert dargestellt, die Ergebnisse der Untersuchungen werden präsentiert und ausgewertet. Darüber hinaus wird die Auswertung diskutiert und im Kontext von Kritik und Limitationen gestellt. Die Ergebnisse werden in Kapitel 5 in Form eines Nutzungskonzepts vorgestellt. Dieses umfasst Hinweise zur Nutzung der Werkzeuge, zu den Einschränkungen, die mit der Nutzung einhergehen, sowie konkrete Best Practices für den Einsatz von LLM-basierten Werkzeugen im SE. Das Kapitel 6 schließt die Arbeit mit einer Zusammenfassung der Ergebnisse, zieht ein Fazit und gibt einen Ausblick auf zukünftige Forschung. In Anhang A hängen weitere Referenzen und Informationen an, welche diese Arbeit vervollständigen.

1.5 ACCSO — ACCELERATED SOLUTIONS GMBH

Diese Arbeit entstand in Zusammenarbeit mit der Accso — Accelerated Solutions GmbH[2]. Accso ist ein modernes und zukunftsorientiertes Unternehmen, welches individuelle IT-Lösungen in enger Zusammenarbeit mit Kunden aus verschiedensten Branchen entwickelt. Hierbei begleitet Accso seine Kunden

2 Homepage von Accso: https://accso.de

über den gesamten Entwicklungsprozess und entwickelt partnerschaftlich Lösungen für komplexe Herausforderungen. Der Fokus liegt dabei nicht nur auf dem Ergebnis, sondern ebenso auf der Art der Zusammenarbeit. Accso steht für eine flexible und transparente Arbeitskultur, in welcher vielfältige Teams ihre gemeinsame Expertise zur Lösung anspruchsvoller Herausforderungen einsetzen. Dafür setzt Accso auf den Einsatz aktueller Technologien und evaluiert kontinuierlich die neusten Trends im Software Engineering. Ursprünglich wurde Accso 2010 in Darmstadt gegründet und zählt inzwischen 260 Mitarbeiter:innen an vier Standorten innerhalb Deutschlands und einem in Kapstadt, Südafrika [Sol].

Die Zusammenarbeit mit Accso kam über Prof. Dr. Markus Voß zustande, den Referenten dieser Masterthesis und Geschäftsführer von Accso. Prof. Voß ist neben seiner Tätigkeit bei Accso seit 2011 als Professor an der Hochschule Darmstadt tätig. Dort lehrt er Module zu den Themen Software-Architektur und Service-Orientierter Architektur.

Im Rahmen dieser Kooperation stehe ich, der Autor, für die Dauer meiner Thesis als Masterand bei Accso mit 16 Wochenstunden unter Vertrag. Ich teilte meine Ergebnisse mit Accso und arbeitete eng mit Experten der Firma zusammen. Für die Pilotuntersuchungen erprobten 19 Software Engineers von Accso die zuvor identifizierten Werkzeuge in Rahmen ihrer täglichen Arbeit. Ich unterstützte dabei durch die Recherche nach LLM-basierten Werkzeugen, durch Vorbereitung und Moderation der Untersuchungen und durch das Zusammentragen von Ergebnissen in Form von Experteninterviews mit den Engineers. Darüber hinaus war ich an der Planung weiterer Events zu diesem Thema beteiligt und erprobte eigenständig in einem Projekt der Firma die Werkzeuge. Die Einblicke und empirischen Erfahrungsberichte in realen Projektszenarien waren entscheidend für diese Arbeit.

1.6 EIN HINWEIS BEZÜGLICH INKLUSIVER SPRACHE

Inwiefern unsere Sprache alle Mitglieder der Gesellschaft inkludiert, ist ein aktuell viel diskutiertes und umstrittenes Thema [Kip23; MS21; Die18]. Im deutschen Sprachgebrauch steht dabei oft das sogenannte generische Maskulinum im Zentrum der Auseinandersetzung. Von diesem ist die Rede, wenn etwa von *Softwareentwicklern* in männlicher Form des Wortes gesprochen wird, jedoch implizit alle Personen jeglichen Geschlechts gemeint sind, solange diese Software entwickeln [MS21]. Es bestehen Ansätze, das generische Maskulinum durch Formulierungen abzulösen, die auf explizitere Weise alle gemeinten Personen einschließen. Beispiele hierfür sind etwa die Verschmelzung der männlichen und weiblichen Formen (*Softwareentwickler:innen*) oder die Verwendung der geschlechterneutralen Form (*Softwareentwickelnde*). Um die Notwendigkeit inklusiver Sprache herrscht Uneinigkeit, obwohl es zumindest Indizien für mehr Geschlechtergleichheit in Ländern gibt, in welchen geschlechterneutrale Sprachen gesprochen werden [PFCL12].

Als Autor dieser Thesis finde ich es vollkommen nachvollziehbar, wenn nicht-männliche Personen sich durch die Verwendung des generischen Mas-

kulinums außen vor gelassen fühlen. Mir ist es wichtig, sämtliche Mitglieder der Gesellschaft in allen Aspekten des Lebens zu inkludieren. Dennoch verwendet diese Arbeit nur selten inklusive Sprache. Stattdessen wird zugunsten der Verständlichkeit auf sperrige, inklusive Formulierungen verzichtet und stattdessen alternierend von verschiedenen Geschlechtern gesprochen oder die neutrale englische Berufsbezeichnung „Software Engineer“ benutzt. Ist im Verlauf dieser Arbeit ein Aspekt anhand eines Beispiels illustriert, in welchem von „einer Engineer und ihrem Workflow“ die Rede, so lässt sich dieses Beispiel auf jegliche Personen verallgemeinern. Dabei wird keinerlei Einschränkung bezüglich Geschlecht, Ethnie, Alter oder andere Attribute der betroffenen Person gemacht.

2 GRUNDLAGEN

Diese Arbeit untersucht den Einsatz KI-basierter Werkzeuge in der Softwareentwicklung. Damit sind Werkzeuge gemeint, welche auf großen Sprachmodellen, auch LLMs genannt, basieren. Bei großen Sprachmodellen handelt es sich um hoch-spezialisierte künstliche neurale Netze, welche der Transformer-Architektur folgen. Künstliche neurale Netze gehören zum Ansatz des maschinellen Lernens, welches wiederum ein Teilgebiet der künstlichen Intelligenz ist. Zudem gehören Sprachmodelle in das Gebiet der Verarbeitung natürlicher Sprache (englisch: Natural Language Processing; kurz NLP), ein Anwendungsgebiet von KI. Auch KI selbst ist nicht trivial zu definieren.

Dieses Kapitel geht auf alle diese Fachbegriffe ein und schafft so eine Grundlage für die vorliegende Arbeit. Zuerst wird in Abschnitt 2.1 der Begriff der künstlichen Intelligenz betrachtet. Darin wird über die Anwendungsgebiete und verschiedene Ansätze der KI gesprochen sowie eine kurze historische Einordnung gegeben. Anschließend wird in Abschnitt 2.2 auf das Anwendungsgebiet der Verarbeitung natürlicher Sprache eingegangen. Es wird auf die Herausforderung der Verarbeitung natürlicher Sprache eingegangen und ebenfalls eine knappe historische Einordnung gegeben. Danach stellt Abschnitt 2.3 den Ansatz des maschinellen Lernens (ML) vor. Dort wird auf künstliche neurale Netze, Deep Learning und die Limitationen des ML eingegangen. Abschnitt 2.4 präsentiert die großen Sprachmodelle, geht kurz auf deren Entstehung und technischen Hintergründe ein und diskutiert deren Kompetenzen und Limitationen. Zuletzt führt Abschnitt 2.5 den Stand der Forschung zu KI-gestütztem Software Engineering auf. Im Nachfolgenden werden die hier untersuchten Werkzeuge auch als KI-Werkzeuge beziehungsweise LLM-basierte Werkzeuge bezeichnet. Alle diese Begriffe stehen für dieselbe Gruppe von Werkzeugen.

2.1 KÜNSTLICHE INTELLIGENZ

Künstliche Intelligenz (kurz: KI) ist ein zentraler Begriff dieser Arbeit. Dieser Abschnitt geht auf eine Begriffsklärung des Begriffs selbst ein, gibt einen Überblick über die Anwendungsgebiete der KI, stellt mit der wissensbasierten KI und dem maschinellen Lernen zwei grundlegende Ansätze gegenüber und gibt eine kurze historische Einordnung.

2.1.1 *Begriffsklärung künstliche Intelligenz*

Der Begriff „künstliche Intelligenz“ ist nicht eindeutig definiert. In der öffentlichen Diskussion wird der Begriff häufig als Synonym für konkrete Anwen-

dungen oder Dienste verwendet. So wird ChatGPT beispielweise häufig als „eine künstliche Intelligenz“ bezeichnet:

> *„Die Entwickler selbst warnen davor, dass die künstliche Intelligenz noch nicht ausgereift genug ist …“* [Mdr]

> *„Die KI lieferte die Antworten* […] *in Schriftform.“* [Enn]

Eine wissensschaftsnähere Betrachtung stammt aus der Encyclopaedia Britannica [Cop] und wird von Humm in dessen Buch *Applied Artificial Intelligence* aufgegriffen. Übersetzt aus dem Englischen lautet sie wie folgt:

> *„künstliche Intelligenz (KI), die Fähigkeit eines digitalen Computers oder eines computergesteuerten Roboters, Aufgaben auszuführen, die üblicherweise intelligenten Wesen zugeschrieben werden.“* [Hum22, S. 2]

Humm weist dabei auf folgende Nuance hin: Die Definition impliziert weder, dass KI-basierte Anwendungen intelligent sind, noch zieht sie einen Vergleich zu menschlicher Intelligenz. Es geht schlicht um die Kompetenz, bestimmte Aufgaben auszuführen. Beispiele seien die Wahrnehmung bzw. Erkennung der eigenen Umwelt, intellektuelle Herausforderungen wie das Lernen, das Abrufen von Wissen oder das Schlussfolgern. Genauso zählten dazu die Kommunikation oder das gezielte Handeln. Eine Gemeinsamkeit dieser Aufgaben sticht hervor: Sie beinhalten eine gewisse Komplexität und lassen sich nicht durch einfachen Handlungsanweisungen lösen — es gibt keine simple Schritt-für-Schritt-Anleitung für das Wahrnehmen der eigenen Umwelt. Stattdessen scheint die erfolgreiche Bearbeitung dieser Aufgaben eine Art von Intelligenz zu erfordern. Solange ein Computerprogramm eine dieser komplexen Aufgaben erfolgreich ausführen kann, gilt es nach dieser Definition als künstlich intelligent. Doch ein Teil dieser Definition ist ambivalent: Die Formulierung „die üblicherweise […] zugeschrieben werden“ trifft keine klare Aussage darüber, wer die Zuschreibung vornimmt. Demnach könnte die Definition von KI je nach Betrachter oder Zeitpunkt unterschiedlich ausgelegt werden, ohne dass sich deren Wortlaut ändert.

Eine alternative Definition liefert Chowdhary in dessen Buch *Fundamentals of Artificial Intelligence*. Er betrachtet künstliche Intelligenz als ein Teilgebiet der Informatik, welches sich mit der „Automatisierung von intelligentem Verhalten“ [Cho20, S. 1] befasst. Intelligenz selbst setze sich aus drei Komponenten zusammen: Wahrnehmung, Analyse und Reaktion. Beispiele für Intelligenz seien die Fähigkeit zu lernen, die Fähigkeit sich an neue Gegebenheiten anzupassen, oder die Fähigkeit gezielt zu planen und zu handeln. Nach Chowdhary ist es das Ziel von KI, die Wesenszüge von Intelligenz grundlegend und mithilfe von Computern zu verstehen. Dabei liege das Augenmerk auf schwierigen Fragestellungen, welche die Natur der menschlichen Kreativität und Intuition oder die Voraussetzungen für Ichbewusstsein betreffen. KI diene hierbei als Werkzeug und Versuchsumfeld, in welchem Theorien über Intelligenz praktisch (via Ausführung in Computerprogrammen) erprobt werden können.

Beide Definitionen stützen sich auf den Begriff des intelligenten Verhaltens. Sie stimmen darin überein, dass intelligentes Verhalten komplex und schwer exakt zu definieren ist, doch beziehen sie eine unterschiedliche Perspektive auf KI: Während künstliche Intelligenz nach Humm nicht zwangsläufig intelligent sein muss, sieht Chowdhary sie als ein Streben danach, „echte“ Intelligenz zu erforschen und zu simulieren. Eine differenzierte Diskussion dieser Perspektiven wirft komplexe philosophische Fragen nach der Natur von Intelligenz auf, die den Fokus dieser Arbeit verlassen. Stattdessen wird hier auf Basis beider Definitionen ein Vorschlag zur Interpretation des Begriffs unterbreitet:

> Künstliche Intelligenz ist ein Teilgebiet der Informatik, welches sich mit der Automatisierung von intelligentem Verhalten beschäftigt. Als intelligentes Verhalten gilt solches, dass üblicherweise mit intelligenten Wesen assoziiert wird. Dieses Verhalten ist komplex, nicht durch einfache Handlungsanweisungen lösbar und erfordert eine Art von Intelligenz zur erfolgreichen Ausführung.

Diese Definition kombiniert Chowdharys Perspektive auf KI als eine Disziplin der Informatik mit Humms Bezeichnung von intelligentem Verhalten. Sie wird durch Chowdharys Anteil greifbarer, indem sie KI als Disziplin und nicht als Fähigkeit definiert. Sie erbt die Ambivalenz von Humms Definition, indem sie die Assoziation von intelligentem Verhalten offen lässt. Dadurch vermeidet sie die Notwendigkeit einer komplexen Definition von Intelligenz, welche unter Umständen gar nicht abschließend möglich ist. Im weiteren Verlauf dieser Arbeit wird von der hier vorgeschlagenen Definition von künstlicher Intelligenz ausgegangen. Die im Rahmen der Untersuchungen betrachteten Werkzeuge, ChatGPT und GitHub Copilot, fallen eindeutig in den Bereich der künstlichen Intelligenz, da diese Werkzeuge auf Informationstechnologie basieren und dabei helfen, intelligentes Verhalten zu automatisieren.

2.1.2 *Anwendungsgebiete von KI*

Als ein weitläufiges Teilgebiet der Informatik beinhaltet die künstliche Intelligenz eine Vielzahl unterschiedlicher Anwendungsgebiete, Methodiken und Technologien. Die Anwendungsgebiete decken viele der oben erwähnten, üblicherweise mit intelligenten Wesen assoziierte, Fähigkeiten ab. Dazu zählen unter anderem [Hum22, S. 4]:

- Schlussfolgern: Darunter fallen etwa logische Programmierung, probabilistisches Schließen und die Verarbeitung komplexer Ereignisse.
- Handeln: beispielsweise Planung, Agententechnologie und Robotik
- Wissen und Wissensrepräsentation
- Lernen: darunter Maschinelles Lernen (ML), Informationsbeschaffung und Data-Mining

- Wahrnehmung: Darunter fallen etwa Computer Vision und Sensorik.
- Kommunikation und die Verarbeitung natürlicher Sprache (englisch: Natural Language Processing, kurz NLP)

Im Kontext dieser Arbeit steht das Anwendungsgebiet der Verarbeitung natürlicher Sprache (NLP) im Vordergrund, da die untersuchten Werkzeuge auf diesem Gebiet arbeiten. Bei „natürlicher Sprache" handelt es sich um Sprache, die Menschen untereinander gebrauchen — wie etwa Vietnamesisch, Swahili, Deutsch oder Französisch. Die Verarbeitung natürlicher Sprachen ist eine komplexe Aufgabe. Sie wird in Abschnitt 2.2 näher beschrieben.

2.1.3 *Wissensbasierte KI gegenüber maschinellem Lernen*

Neben der Differenzierung zwischen unterschiedlichen Anwendungsgebieten der KI, lässt sich die künstliche Intelligenz in zwei fundamental unterschiedliche Ansätze gliedern: *wissensbasierte KI* — auch symbolische KI genannt — und *maschinelles Lernen* — auch subsymbolische oder nicht-symbolische KI genannt.

In der wissensbasierten KI wird Wissen explizit und menschenlesbar dargestellt. Das geschieht meist in Form von Regeln, Graphen oder semantischen Netzen. Auf Basis dieses strukturieren Wissens ist es möglich Schlussfolgerungen zu ziehen, also neues implizites Wissen aus bereits Vorhandenem abzuleiten, um komplexe Fragen zu beantworten [Hum22, S. 4-5]. Einen gänzlich anderen Ansatz verfolgt das maschinelle Lernen. Hierbei wird Wissen nicht explizit beschrieben, sondern impliziert über eine Trainingsphase gewissermaßen „erlernt". Als Ausgangsbasis dient eine große Menge an Trainingsdaten, mithilfe derer ein sogenanntes Modell trainiert wird. Nach der Trainingsphase ist dieses Wissen durch eine riesige Menge an Zahlenwerten in dem Modell abgebildet und kann nicht mehr von Menschen nachvollzogen werden. Das Modell ist jetzt in der Lage dazu, das „erlernte" Wissen auf Situationen ähnlich dem Trainingsdatensatz anzuwenden, ohne dass exakt diese Situationen Teil des Trainings waren [Hum22, S. 4-5]. Warum „erlernt" hier in Anführungszeichen steht, wird in Unterabschnitt 2.3.4 besprochen.

Ein Beispiel für einen Einsatz wissensbasierter KI ist ein Expertensystem für klassische Kunst. Hierbei repräsentiert etwa ein semantisches Netz Fakten und Zusammenhänge über Kunstwerke, deren Künstler und weiteres. Dieses Wissen wird vorab von Domänenexperten bereitgestellt und überprüft. Zur Laufzeit kann das System anhand der Fakten und Zusammenhänge Schlussfolgerungen ziehen und Fragen beantworten — sogar solche, welche nur implizit im semantischen Netz repräsentiert sind. Ein Beispiel für einen Einsatz von maschinellem Lernen ist die Erkennung von Spam E-Mails. Dabei wird ein Modell mit einer großen Menge an E-Mail trainiert, welche vorab in die Kategorien Spam und Nicht-Spam vorsortiert sind. Im Anschluss an das Training ist das Modell in der Lage, neue E-Mails als Spam oder Nicht-Spam zu klassifizieren. Abbildung 2.1 stellt das Beispiel des Spamfilters grafisch dar.

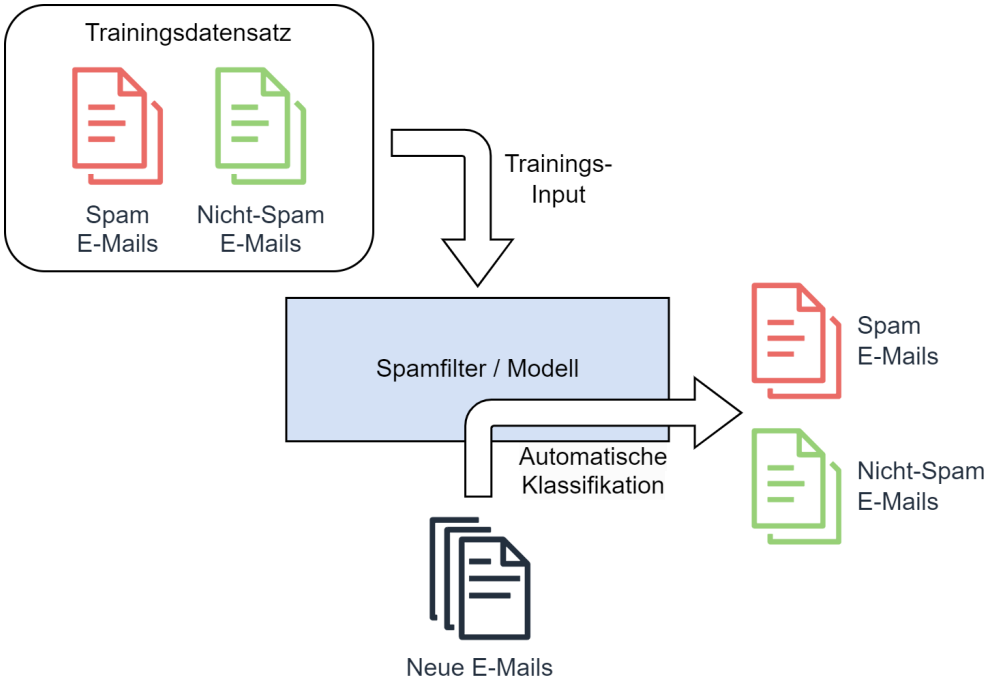


Abbildung 2.1: Illustration der Funktionsweise eines Spam-Filters auf Basis von maschinellem Lernen. Die Abbildung ist inspiriert von [Hum22, Fig. 2.1: Spam filtering].

Die Stärken und Schwächen beider Ansätze greifen ineinander und ergänzen sich nahezu perfekt. Bei wissensbasierter KI ist es beispielsweise aufgrund der menschenlesbaren Darstellung möglich, die Korrektheit durch Domänenexperten zu überprüfen, während Fehler im implizit „erlernten“ Wissen des ML unvermeidbar sind und nicht von Menschen nachvollzogen oder korrigiert werden können. Dafür eignet sich wissensbasierte KI ausschließlich für den Einsatz in Domänen, in welchen Wissen explizit dargestellt werden kann — wie etwa Expertensysteme. Sobald das Wissen nicht explizit beschreibbar ist — etwa für das Laufen auf zwei Beinen oder bei der Erkennung von Spam E-Mails — eignet sich maschinelles Lernen hingegen viel eher zur Bewältigung dieser Aufgabe. Tatsächlich sollte maschinelles Lernen niemals verwendet werden, insofern eine konkrete beschreibbare Vorgehensweise bekannt ist, um ein Problem zu lösen. Die Abhängigkeit von Trainingsdaten ist ebenfalls ein Unterschied. Während wissensbasierte Systeme auch in Domänen mit wenigen expliziten Beispielen einsetzbar sind, benötigt maschinelles Lernen oft viele tausenden vorsortierte Beispieldatensätze, um effektiv eingesetzt werden zu können. Aufgrund ihrer großen Verschiedenheiten sind die Einsatzgebiete der Ansätze sehr unterschiedlich. Historisch betrachtet war wissensbasierte KI, während den Anfängen der KI-Forschung ab den 1950er Jahren, der deutlich populärere Ansatz — auch da die verfügbare Rechenleistung damals den Anforderungen des rechenaufwandshungrigen ML nicht genügte. Inzwischen hat sich diese Präferenz umgekehrt, sodass heute beim Thema KI vor allem an das maschinelle Lernen gedacht wird. Es ist denkbar, dass Hy-

bridansätze, welche die Stärken beider Ansätze kombinieren, in der Zukunft an Relevanz gewinnen [Hum22, S. 4-5, 12–13].

Im Kontext dieser Arbeit liegt der Fokus auf dem Ansatz des maschinellen Lernens, da die untersuchten Werkzeuge auf diesem Ansatz basieren. Abschnitt 2.3 geht genauer auf das ML, dessen Konzepte und Limitationen ein.

2.1.4 *Eine kurze Geschichte der künstlichen Intelligenz*

Dieser Unterabschnitt gibt einen kurzen Überblick über die Geschichte der künstlichen Intelligenz. Die Idee intelligenter Maschinen reicht weit in die Vergangenheit zurück. Ein historisches Beispiel ist der sogenannte „Schachtürke" — eine Schachmaschine, die im 18. Jahrhundert in Europa auftrat. Ihr Erfinder ließ bei Zuschauern den Eindruck entstehen, sie könne durch einen mechanischen Trick selbstständig Schach spielen. Tatsächlich war darin jedoch ein menschlicher Schachspieler versteckt, welcher die Maschine bediente [Wik23c]. Die eigentliche Historie der künstlichen Intelligenz beginnt jedoch erst im 20. Jahrhundert.

Den Grundstein für die Forschung an künstlicher Intelligenz legte Alan Turing (1912 - 1954) durch seine Arbeit auf dem Gebiet der Berechenbarkeitstheorie und durch die Entwicklung seines sogenannten Turing Test. Dieser prüfte die Kompetenz eines Computers, menschliches Verhalten so gut zu imitieren, dass die Imitation nicht mehr von echtem menschlichem Verhalten zu unterscheiden war. Der Begriff „künstliche Intelligenz" kam jedoch erst zwei Jahre nach Turings Tod auf. John McCarthy gilt als derjenige, der den Begriff KI im Rahmen des Dartmouth Workshop 1956 erstmals verwendete. Die darauf folgenden zwei Jahrzehnte, 1960er bis in die 1980er Jahre, waren von einem massiven KI-Hype geprägt. Dieser Hype wurde durch utopische Zukunftsprognosen angefeuert [Hum22, S. 5]. Beispiele sind:

> *„Innerhalb von 10 Jahren werden Computer uns nicht einmal mehr als Haustiere halten."* — Marvin Minsky, 1970, aus dem Englischen übersetzt.

> *„Maschinen werden in der Lage sein, jede Aufgabe zu erledigen, die ein Mensch tun kann."* — Herbert Simon, 1985, aus dem Englischen übersetzt.

Gerade in den USA sorgte der Hype für eine großangelegte finanzielle Förderung von KI-Projekten. Doch als schließlich die Ergebnisse hinter den aufgebauschten Erwartungen zurückblieben, fielen die Luftschlösser in sich zusammen. Selbst die hochentwickeltsten KI-Systeme scheiterten an Aufgaben, die von kleinen Kindern mühelos erledigt wurden. Die Desillusionierung führte zu einer enormen Kürzung der Gelder und einem Zusammenbruch der KI-Bubble. Dieser Zeitraum, die 1980er bis 1990er Jahre, wird deshalb auch der KI-Winter genannt [Hum22, S. 6].

Nach dem KI-Winter begann sich das Forschungsfeld langsam und im Stillen zu erholen. Mit der Entwicklung vieler neuer Methoden und Technologien sowie zunehmender Verfügbarkeit kostengünstiger Rechenkapazitäten

begann ein neuer Hype, welcher bis in die Gegenwart anhält [Hum22, S. 6] und heute, im Jahre 2023, die Welt fest im Griff zu haben scheint.

2.2 NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) ist das Anwendungsgebiet der künstlichen Intelligenz, welches im Rahmen dieser Arbeit im Fokus steht. Es beschäftigt sich mit der Verarbeitung von sogenannter „natürlicher Sprache". Natürliche Sprachen sind solche, die Menschen untereinander gebrauchen, wie zum Beispiel Vietnamesisch, Swahili, Deutsch oder Französisch. Im Zeitalter von Internet, Smartphones und Smart Home ist die Verarbeitung natürlicher Sprachen längst im Alltag angekommen. Zu den Aufgabengebieten[1] zählen unter anderem:

- Natürliche Sprachen ineinander übersetzen; z.B. von Griechisch nach Spanisch via Übersetzungsapp
- Die Rechtschreibung eines Texts überprüfen; z.B. in einem Texteditor
- Gesprochene Worte verstehen und in Buchstaben übersetzen; z.B. die Diktat-Funktion des Smartphones
- Texte in gesprochene Sprache übersetzen und laut vorlesen; z.B. Barrierefreiheitsfunktionen in Webbrowsern
- Fragen in natürlicher Sprache verstehen und beantworten; z.B. im Dialog mit einem Chatbot
- Auf Basis einer textuellen Beschreibung einen Codeabschnitt generieren; z.B. mithilfe eines KI-Pair-Programmers

2.2.1 *Herausforderungen von NLP*

Die Verarbeitung natürlicher Sprache ist eine sehr herausfordernde Aufgabe. Dieser Unterabschnitt stellt Aspekte dieser Herausforderung anhand einiger Beispiele vor.

In natürlichen Sprachen besitzen Wörter mehrere Bedeutungen, wie etwa *Bank* (Sitzgelegenheit oder Finanzinstitut) oder *Linse* (Hülsenfrucht oder optisches Bauteil), deren tatsächliche Bedeutung von dem Kontext abhängt. Relevant ist sowohl der Kontext des umliegenden Textes, als auch gesellschaftliches Allgemeinwissen. Ist beispielsweise von der kaputten Linse einer Kamera die Rede, ist in diesem Fall wahrscheinlich keine Hülsenfrucht zu Schaden gekommen, da das Adjektiv „kaputt" im Kontext des umliegenden Textes eher zu einem optischen Bauteil passt. Um folgenden Satz korrekt einzuordnen, ist Allgemeinwissen notwendig: *Ich habe ein Selfie mit Scooter gemacht!* Die deutsche Musikgruppe „Scooter" zu kennen, eröffnet neben einem Foto eines

1 Diese Beispiele stammen aus [Hum22, S. 99].

Fahrzeugs eine weitere Interpretationsmöglichkeit. Trotzdem gibt es an dieser Stelle keine zu 100% korrekte Antwort. Beide Interpretationen sind valide. Auch das ist eine Herausforderung beim Verständnis natürlicher Sprache.

Des Weiteren besitzen nicht alle Wörter in einem Satz die gleiche Relevanz bei dessen Interpretation. Im Beispiel *Ich habe nach dem Konzert Backstage ein Selfie mit Scooter gemacht!* sind die Worte „Konzert", „Backstage" und „Scooter" weitaus aussagekräftiger als „habe", „nach" oder „dem". Mit Fokus auf diese Worte erscheint die Interpretation eines Fotos mit der Musikgruppe deutlich wahrscheinlicher als das eines Fahrzeugs.

Auch sogenannte Koreferenzen sind eine nicht-triviale Herausforderung. Dabei verweisen verschiedene Worte auf dieselbe Entität, wie zum Beispiel in: *Dennis gab mir einen Apfel. Er schmeckte gut.*, wobei sowohl „Apfel", als auch „er" sich auf dieselbe Frucht beziehen. Weitere Herausforderungen sind unter anderem die Analyse von Grammatik oder Satzbau sowie Fehler in diesen, welche die Bedeutung eines Satzes komplett verändern können. Handelt es sich bei *Komm, wir essen Opa* um einen Aufruf zum Kannibalismus oder wurde schlicht ein Komma vor dem letzten Wort vergessen? Diese Beispiele mögen aus der Perspektive eines Menschen trivial wirken, doch ist die computerbasierte Verarbeitung mittels vorprogrammierter Wenn-Dann-Sonst-Regeln eine monumentale Herausforderung.

2.2.2 *Eine kurze Historie der NLP*

Für die Informatik war die Verarbeitung natürlicher Sprache historische betrachtet eine sehr herausfordernde Aufgabe. Insbesondere für herkömmliche Computerprogramme, welche nicht auf maschinellem Lernen basieren, stellt die Komplexität von natürlicher Sprache eine kaum zu bewältigende Herausforderung dar. Erst durch die seit 2020 aufgekommenen sogenannten großen Sprachmodelle (LLMs) scheint es möglich, die Herausforderung NLP zu bewältigen. Die Funktionsweise von LLMs wird in Abschnitt 2.4 erläutert. Der vorliegende Unterabschnitt beschreibt kurz die Historie der NLP bis 2020. Dabei lässt sich die Geschichte der NLP in fünf Phasen einteilen.

Phase 1 — 1940er-1960er Jahre

In den Anfängen wurde sich vor allem aus linguistischer Perspektive mit Syntax und Semantik befasst. Trotz der Entwicklung erster Formalismen, Werkzeuge und Ideen, entstanden keine Systeme von signifikanter Qualität oder signifikantem Umfang. Den Begriff KI als solches gab es noch nicht — gerade da der Begriff erst 1956 geprägt wurde [Jon94, S. 5-6]. Dennoch wurden bereits wichtige Grundsteine gelegt: So generierte Claude Shannon bereits 1948 den ersten Text mithilfe von Wahrscheinlichkeiten von aufeinander folgenden Wörtern [Sha48, S. 7] — ein Ansatz, der heute noch große Relevanz besitzt. Generell versuchten die Forscher in dieser Phase „mit dem Kopf die Wand sehr harter Probleme zu durchbrechen" [Jon94, S. 6].

Phase 2 — 1960er-1970er Jahre

In dieser Phase verschob sich der Ansatz auf Allgemeinwissen, Wissensrepräsentation. Linguistik wurde durch wissensbasierte KI abgelöst. Zunächst herrscht großer Optimismus bezüglicher der erzielten Fortschritte, doch gegen Ende dieser Phase wurde realisiert, dass bereits die einfachsten Aufgaben der NLP weitaus schwieriger waren, als zunächst angenommen [Jon94, S. 6-8].

Phase 3 — 1970er-1980er Jahre

Als Reaktion auf die Fehlschläge wurde sich wieder an die Linguistik gewand. Grammatik und Logik rückten erneut in den Fokus, gemeinsam mit einer Verfeinerung der wissensbasierten KI. Die dritte Phase war geprägt vom Aufkommen der Prädikatenlogik und der Popularität von probabilistischen Netzen. Die Forschung beschäftigte sich erstmalig ernsthaft mit der Generierung von mehrzeiligem Text [Jon94, S. 8-10]. Die Entwicklung des Backpropagation Learning Algorithm fällt ebenfalls in diese Phase [Fra20, S. 1387] — ein Ansatz, der heute noch hochrelevant ist.

Phase 4 — Ende 1980er Jahre bis 2000

In dieser Phase gewannen neben wissensbasierter KI auch statistische und datenbasierte Ansätze an Bedeutung, aufgrund der erhöhten Verfügbarkeit von maschinenlesbaren Datensätzen und Wörterbüchern. Das Erzielen großer Fortschritte hatte eine erneute Inangriffnahme der Herausforderungen von NLP zur Folge. Der technische Fortschritt und die Dekaden an Erfahrung ermöglichten die Konstruktion erster kompetenter Systeme — auch wenn diese Systeme noch nicht ausgereift und auf den Einsatz in bestimmten Domänen beschränkt waren [Jon94, S. 10-12].

Phase 5 — 2000er Jahre bis heute (2023)

Die fünfte Phase ist vom Einsatz des maschinellen Lernens dominiert. Während das Interesse für die Erforschung von ML auch außerhalb des Teilgebietes NLP bereits in den 1990ern wuchs, ermöglichten erst Durchbrüche in drei Trends den Einsatz von maschinellem Lernen. Der erste Trend ist „Big Data“. Die zunehmende Sammlung enormer Datenmengen trieb die Entwicklung von ML mehr aus praktischer Notwendigkeit als aus Neugierde voran. Der zweite Trend ist eine signifikante Kostenreduzierung für parallele Datenverarbeitung und Hauptspeicher, welche die Arbeit auf den enormen Datenmengen überhaupt erst wirtschaftlich machte. Der dritte Trend ist die Entwicklung neuer Algorithmen im Bereich Deep Neural Learning (DNL) [Fra20, S. 1387-1388]. Deep Neural Learning ist eine spezielle Form des maschinellen Lernens, welche in Unterabschnitt 2.3.3 näher beschrieben ist.

Alles in allem zeigt die Historie von NLP deutlich, wie herausfordernd dieses Thema ist. Über viele Dekaden hinweg erprobten eine große Zahl von Forschern eine Vielzahl von Ansätzen, um die Herausforderungen von NLP zu

bewältigen. Auch wenn diese Arbeit selten konkrete Erfolge vorzuweisen hatte, legten sie an vielen Stellen den Grundstein für die heutigen Erfolge.

2.3 MASCHINELLES LERNEN

Da die Grundzüge des maschinellen Lernens sowie eine Abgrenzung zu wissensbasierter KI bereits in Unterabschnitt 2.1.3 dargestellt sind, geht dieser Abschnitt nur kurz auf die Grundlagen des ML ein. Maschinelles Lernen erlaubt die Bewältigung von Problemen, für welche keine beschreibbare Lösung bekannt ist. Anstatt eine konkrete Lösung zu programmieren, werden sogenannte Modelle auf der Basis von Trainingsdaten automatisch generiert (auch *trainiert* genannt). Die Modelle „lernen" gewissermaßen Muster aus den Trainingsdaten und können diese Muster auf neue Daten anwenden, welche nicht Teil des Trainings waren. Zu den Anwendungsfällen zählen beispielweise die Erkennung von Tieren in Bildern, das Laufen auf zwei Beinen oder die Filterung von Spam E-Mails. In all diesen Fällen ist keine simple Schritt-für-Schritt-Anleitung für die Lösung dieser Aufgaben bekannt — Wie kann auf deterministische Weise ein Frosch von einer Katze unterschieden werden?

2.3.1 *Bereiche, Aufgabengebiete und Ansätze*

Das ML lässt sich in drei Bereiche unterteilen, unter welche jeweils verschiedene Aufgabengebiete fallen. Abbildung 2.2 stellt diese Unterteilung grafisch dar. Diese Darstellung erhebt keinen Anspruch auf Vollständigkeit.

Im Bereich des *Supervised Learning* wird das Modell mithilfe von explizit beschriebenen Trainingsdaten trainiert. Das Ziel ist das Modell so weit zu trainieren, dass es neuen Daten die korrekte Beschreibung zuzuordnen kann. In diesem Bereich wird zwischen den Aufgabengebieten *Klassifizierung* und *Regression* unterschieden. Bei der Klassifizierung besteht der Trainingsdatensatz aus Daten, welche jeweils mit einer Einzelnen aus einer endlichen Zahl von Kategorien beschriftet sind. Das Modell wird darauf trainiert, neue Daten in diese Kategorien einzuordnen. In dieses Aufgabengebiet fällt eines der obigen Beispiele: die Erkennung von Tieren in Bildern. Der Trainingsdatensatz besteht aus einer Vielzahl von Bildern, welche jeweils mit der Art des abgebildeten Tieres beschriftet sind. Nach dem Training kann das Modell neue Bilder in Kategorien wie etwa „Katze", „Hund", „Frosch", etc. einordnen. Im Fall der Regression geht es um die Zuordnung von kontinuierlichen Datensätzen auf kontinuierliche Werte. Etwa um die Vorhersage des Preises eines Hauses auf Basis von dessen Größe, der Anzahl der Zimmer und ähnlicher Parameter [Hum22, S. 15-16].

Das *Unsupervised Learning* zielt auf die Erkennung von Korrelationen in riesigen Datensätzen ab, die von Hand für Menschen nur schwer erkennbar sind. In diesem Bereich werden keine beschrifteten Trainingsdaten verwendet. Stattdessen können diese Korrelationen auf verschiedene Arten detektiert werden. Bei der *Clusterbildung* werden Daten in Gruppen zusammengefasst, in welchen eine Ähnlichkeit besteht, die initial unbekannt war. Etwa das Kauf-

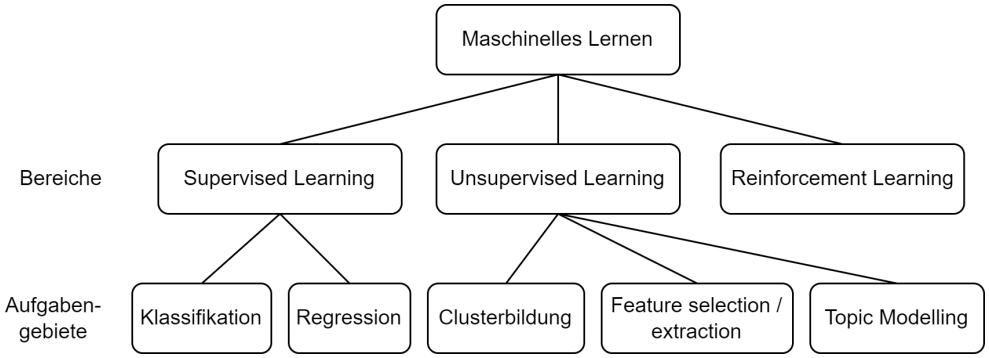


Abbildung 2.2: Illustration der Bereiche und Aufgabengebiete des maschinellen Lernens. Die Abbildung ist übernommen aus [Hum22, Fig. 2.5: ML areas and tasks] und übersetzt.

verhalten verschiedener Kundengruppen. Bei der *Feature selection / extraction* werden die wichtigsten Merkmale eines Datensatzes selektiert bzw. extrahiert. Das Ziel ist die automatische Auswahl von relevanten Merkmalen, um die Komplexität des Datensatzes zu reduzieren, ohne jedoch wichtige Informationen zu verlieren. Dieses Aufgabengebiet kann verwendet werden, um Daten für andere Aufgabengebiete vorzubereiten. Im *Topic Modeling* wird auf Basis textueller Dokumente versucht, Themen zu identifizieren, welche in mehreren Dokumenten auftreten, um die Dokumente entsprechend dieser Einteilung zu sortieren. Ein Beispiel hierfür ist etwa die automatische Extraktion relevanter Fachbegriffe für ein bestimmtes Forschungsfeld [Hum22, S. 16-17].

Das *Reinforcement Learning* ist der Art und Weise wie Menschen lernen am ähnlichsten. In einer dynamischen Umgebung hat ein sogenannter Agent die Aufgabe, ein bestimmtes Ziel zu erreichen. Um die Leistung des Agenten zu verbessern, wird dieser entsprechend einer Metrik für Fehlentscheidungen bestraft und für richtige Entscheidungen belohnt. Der Agent versucht, die Belohnung zu maximieren sowie die Bestrafung zu minimieren und verbessert sich so kontinuierlich. Der Erfolg von Reinforcement Learning hängt im höchsten Maße von der Qualität der verwendeten Metrik ab [Hum22, S. 17].

Das maschinelle Lernen umfasst viele unterschiedliche Ansätze, um den verschiedenen Bereichen und Aufgabengebieten zu begegnen. Dazu zählen etwa *Entscheidungsbäume*, deren Modell aus einer Vielzahl von Entscheidungsknoten und Verzweigungen besteht. Sie können für Klassifikations- und Regressionsaufgaben im Supervised Learning verwendet werden [Hum22, S. 18]. Ein weiterer Ansatz sind *Support Vector Machines*, welche eine Reihe von Methoden aus dem Supervised Learning zusammenfassen. Diese werden verwendet, um in einem hochdimensionalen mathematischen Raum eine Hyperebene aufzustellen, welche die Daten möglichst gut trennt [MY15, S. 950]. Der wohl prominenteste Ansatz im maschinellen Lernen sind jedoch *künstliche neurale Netze* (englisch: Artificial Neural Network (ANN)). Sie werden im nächsten Unterabschnitt im Detail vorgestellt, da die in dieser Arbeit betrachteten Werkzeuge auf großen Sprachmodellen — einer spezialisieren Variante von ANNs — basieren. Große Sprachmodelle im Allgemeinen verwenden so-

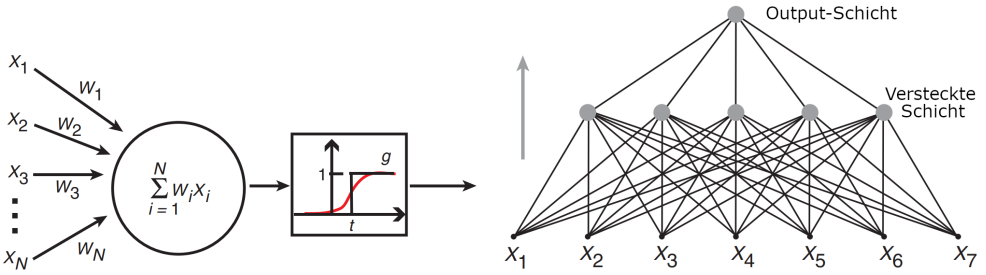
wohl den Bereich des Unsupervised als auch den des Supervised Learning. Im Allgemeinen geschieht ein erster Teil des Trainings — das sogenannte Pre-Training — mittels Unsupervised Learning und ein zweiter Teil — das sogenannte Fine-Tuning — mittels Supervised Learning [Rad+19, S. 1-2ff].

2.3.2 *Künstliche neurale Netzwerke*

Künstliche neurale Netzwerke (ANN) sind inspiriert von der Natur des menschlichen Gehirns. Das menschliche Gehirn und Nervensystem besteht aus einer Vielzahl von miteinander verbundenen Neuronen, welche elektrische Signale durch den Körper übermitteln. Ein Neuron gibt eingehende Signale dabei nur dann weiter, wenn die Summe dieser Signale stark genug ist — also einen bestimmten Schwellwert überschreitet. In diesem Fall wird das Neuron aktiv, „feuert“ und sendet ebenfalls ein Signal weiter, welches potenziell angrenzende Neuronen aktiviert. Wenn ein Neuron wiederholt aufgrund eines eingehenden Signals aktiviert wird, verstärkt sich durch biochemische Prozesse die Verbindung zwischen den beiden Neuronen, sodass das Neuron in Zukunft bereits bei einem niedrigeren Schwellwert feuert. Das Gehirn lernt durch Anpassungen in der Stärke der Verbindungen zwischen den Neuronen. So ändert sich das Verhalten einzelner Neuronen und damit des gesamten Gehirns [Hum22, S. 18-19].

ANN sind eine mathematische Abstraktion dieses Konzepts. Künstliche Neuronen sind über künstliche Synapsen miteinander verbunden. Jede dieser Verbindungen hat ein Gewicht, welches die Stärke der Verbindung repräsentiert. Wie beim menschlichen Gehirn wird ein Neuron aktiviert, wenn die Summe der eingehenden Signale (multipliziert mit dem jeweiligen Gewicht der Synapse) einen bestimmten Schwellwert überschreitet. Die Stärke des ausgehenden Signals wird dabei durch eine mathematische Funktion bestimmt. Abbildung 2.3a zeigt ein künstliches Neuron. Die künstlichen Neuronen sind in Schichten (englisch: *Layers*) angeordnet, wobei die erste Schicht die Eingabedaten erhält und die letzte Schicht die Ausgabe des Netzes darstellt. Alle Schichten zwischen diesen werden als sogenannte versteckte Schichten bezeichnet [Hum22, S. 18-20]. Abbildung 2.3b zeigt ein ANN, welches aus drei Schichten besteht. Ein konkretes Netzwerk mit konkreten Gewichten ist dabei das Modell dieses ML-Ansatzes.

Zu Beginn des Trainings wird das Netz mit zufälligen Gewichten initialisiert. Während des Trainings werden die Trainingsdaten dann durch das Netzwerk geleitet, wobei die Ausgaben des Netzwerks mit den gewünschten, korrekten Ausgaben verglichen werden. (Die Trainingsdaten sind vorab händisch mit den korrekten Ausgaben beschriftet worden.) Es wird der Fehler zwischen der gewünschten und tatsächlichen Ausgabe berechnet und dieser wird mittels des „Back Propagation“-Algorithmus durch das Netzwerk zurückgerechnet. Hierbei werden die Gewichte der Verbindungen zwischen den Neuronen so angepasst, dass der Fehler minimiert wird. Diese Gewichte werden auch als Parameter des Netzes bezeichnet. Dieser Vorgang wird viele Male iterativ wiederholt, bis der Fehler über alle Trainingsdaten hinweg minimal



(a) Ein künstliches Neuron. (b) Ein künstliches neuronales Netzwerk.

Abbildung 2.3: Abbildungen eines künstlichen Neurons und eines ANN. Das Neuron verfügt über die eingehenden Signale x_1 bis x_n, die jeweils mit den Gewichten w_1 bis w_n gewichtet sind. Im Neuron wird die Summe aller Signale gebildet und anhand der Funktion g mit dem Schwellwert t verglichen. Das ANN besteht aus 3 Schichten: Die Input-Schicht umfasst 7 Neuronen, die versteckte Schicht 5 Neuronen und die Output-Schicht 1 Neuron. Die Bilder wurden übernommen aus [Kroo8, Figure 1].

und das Training abgeschlossen ist. Durch diese Minimierung des absoluten Fehlers kann ein Netzwerk zur Lösung des entsprechenden Problems trainiert werden. Die Suche nach dem globalen Minimum des absoluten Fehlers ist dabei keine triviale Aufgabe [Kroo8, S. 196].

Nachdem das Netzwerk trainiert ist, sind die Gewichte innerhalb des Netzes fix. Ab diesem Punkt kann von einem trainierten Modell gesprochen werden. Nun können neue Daten in das Netzwerk gegeben werden, deren korrekte Beschriftungen unbekannt sind und mithilfe des Netzes bestimmt werden sollen. Im Fall der Klassifikation etwa, wird die Ausgabe des Netzes als Wahrscheinlichkeit interpretiert, mit welcher die Eingabe zu einer bestimmten Kategorie gehört. Auf diese Weise können neue Daten automatisiert klassifiziert werden [Hum22, S. 20].

Vor der Klassifikation ist es jedoch wichtig, die Kompetenz des Netzwerkes zu validieren. Hierzu wird gängigerweise ein unabhängiger Datensatz verwendet, welcher nicht Teil des Trainingsdatensatzes war — dem Netz also vollkommen unbekannt ist. Für den Validierungsdatensatz sind ebenfalls die korrekten Beschriftungen bekannt. So kann analog zum Training ein absoluter Fehler über alle Datensätze berechnet werden. Anders als im Training werden jedoch nicht die Gewichte des Netzwerks angepasst. Stattdessen dient der absolute Fehler als Bewertungsmetrik für die Kompetenz des Netzwerks [Kroo8, S. 197]. Sollte die Validierung zeigen, dass die Kompetenz des Netzwerks nicht vom Trainingsdatensatz auf den Validierungsdatensatz übertragbar ist, können Anpassungen vorgenommen werden. Der Erfolg des Trainings hängt dabei von vielen Parameter ab, die angepasst werden können. Darunter fallen etwa die Vorverarbeitung der Trainings- und Validierungsdaten oder die sogenannten Hyperparameter des Netzwerks. Die Hyperparameter definieren die Struktur des Netzes, wie etwa: die Anzahl der versteckten Schichten, der Neuronen pro Schicht, die verwendete Funktionen für Aktivierung und Back

Propagation, die Anzahl der Trainingsiterationen und viele mehr [Hum22, S. 29].

Künstliche neurale Netze illustrieren anschaulich, warum die meisten Ansätze des ML „Blackboxes“ sind, also die Art und Weise auf die deren Ausgaben zustande kommen, nicht von Menschen nachvollzogen werden kann. In der Praxis eingesetze ANNs haben eine Vielzahl von Schichten, die jeweils viele Neuronen enthalten. Die Gesamtzahl der Gewichte innerhalb eines Netzes erreicht auf diese Weise schnell viele Millionen bis hin zu einigen Milliarden. Die Pfade, welche die Eingabe im Netz durchläuft, sind von solch enormer nummerischer Komplexität, dass Menschen diese unmöglich nachvollziehen können.

2.3.3 *Deep Neural Learning*

Bei den großen Sprachmodellen, auf welchen die betrachteten Werkzeuge basieren, handelt es sich um eine spezialisierte Variante künstlicher neuraler Netze, welche zum sogenannten Deep Neural Learning gehören (kurz auch DNL oder Deep Learning). Im Kern geht es dabei um sogenannte *tiefe neurale Netze* (englisch: Deep Neural Networks (DNNs)): ANNs, welche mehr als eine versteckte Schicht und dadurch um ein Vielfaches mehr Neuronen besitzen. Diese Komplexität erlaubt es ihnen, komplexe Hierarchien von Konzepten abzubilden und damit im Vergleich eine enorm gesteigerte Kompetenz zu erreichen [Nie15, S. 37].

Vergleichbar ist diese Abbildung von Hierarchien von Konzepten mit der Zerlegung eines komplexen Problems in weniger komplexe Teilprobleme. Im Kontext eines künstlichen neuralen Netzwerks würden diese Teilprobleme jeweils von Sub-Netzwerken gelöst werden. Die Ausgaben dieser Sub-Netzwerke würden dann in einem letzten Schritt verwendet werden, um das ursprüngliche Problem zu lösen. Abbildung 2.4 zeigt eine Visualisierung dieser Idee. Dabei wird die komplexe Aufgabe, zu erkennen, ob ein Bild ein Gesicht enthält, in die Teilprobleme *Erkennung eines linken Auges*, *Erkennung einer Nase*, und so weiter zerlegt. Diese Teilaufgaben fallen jeweils Sub-Netzwerken zu, deren Ausgaben in einem letzten Schritt verwendet wird, um das ursprüngliche Problem zu lösen [Nie15, S. 35-37].

Tiefe neurale Netze umfassen in der Praxis eine sehr große Zahl an Schichten und Neuronen. Übertragen auf das Beispiel bedeutet das, sich nicht nur auf Sub-Netzwerke zu beschränken. Wie in Abbildung 2.5 dargestellt, könnten die Sub-Netzwerke jeweils selbst wieder in Subsub-Netzwerke unterteilt werden, um die möglicherweise immer noch sehr komplexen Teilprobleme weiter aufzuspalten und zu vereinfachen. Hierbei wird etwa die Erkennung eines linken Auges in die *Erkennung einer Augenbraue*, die *Erkennung von Wimpern* und so weiter zerlegt [Nie15, S. 36].

Das gezeigte Beispiel ist jedoch nur eine Analogie zum Deep Learning. Im DNL findet generell keine tatsächliche Aufteilung in Subnetzwerke statt. Diese wird auch nicht, wie hier angedeutet, vom Menschen entworfen und ist auch nicht durch Menschen nachvollziehbar. Stattdessen werden diese Strukturen

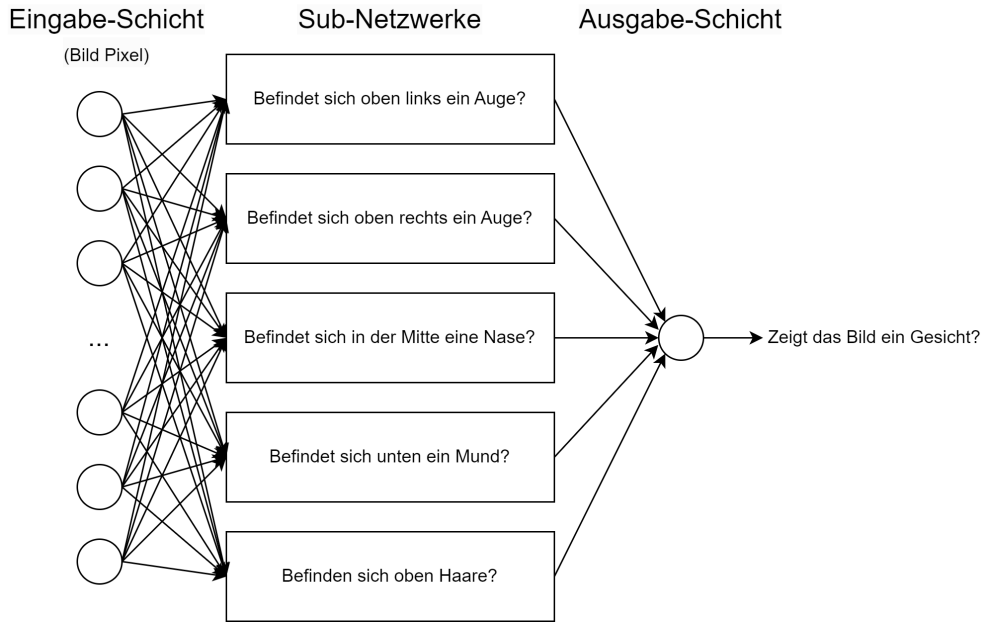


Abbildung 2.4: Eine Visualisierung des Konzeptes hinter Deep Learning: eine Zerlegung eines komplexen Problems in weniger komplexe Teilprobleme, welche jeweils von Sub-Netzwerken gelöst werden. Siehe Unterabschnitt 2.3.3 für mehr Details. Die Abbildung ist übernommen aus [Nie15, S. 36] und übersetzt.

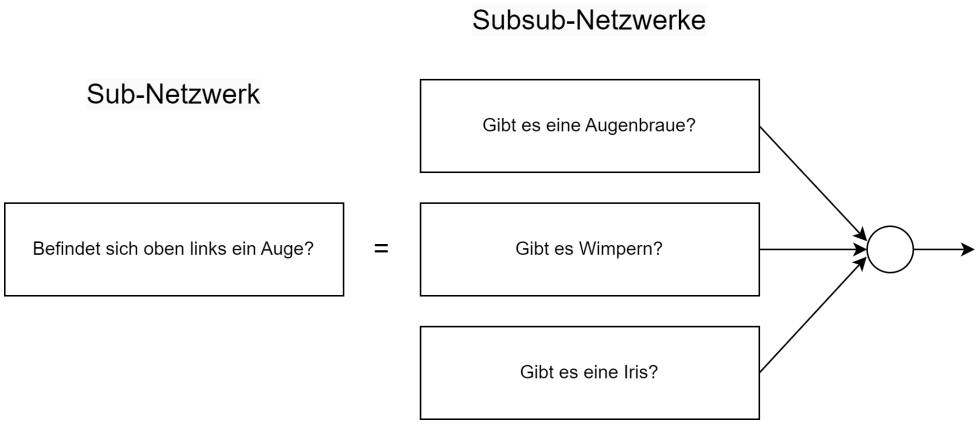


Abbildung 2.5: Eine weitere Visualisierung des Konzeptes hinter Deep Learning. Die Sub-Netzwerke können jeweils selbst weiter in Subsub-Netzwerke unterteilt werden. Siehe Unterabschnitt 2.3.3 für mehr Details. Die Abbildung ist übernommen aus [Nie15, S. 36] und übersetzt.

genau wie die Gewichte der ANNs im Rahmen eines Trainings „erlernt“ [Nie15, S. 37]. So kommt die Abbildung der komplexen Hierarchien von Konzepten in den Modellen und die damit einhergehende enorme Kompetenzsteigerung zustande [Nie15, S. 37].

Eine Spezialisierung von Deep Learning sind die großen Sprachmodelle, auf welchen die in dieser Arbeit betrachteten Werkzeuge basieren. Diese werden in Abschnitt 2.4 genauer betrachtet.

2.3.4 *Weitere Limitationen von ML*

Maschinelles Lernen ist ein effektiver Ansatz zum Lösen von Problemen, für welche keine beschreibbare Lösung bekannt ist. Doch hat ML über die bereits oben erwähnte Nicht-Nachvollziehbarkeit und der Abhängigkeit einer großen Menge an Trainingsdaten auch weitere Limitationen, welche in diesem Unterabschnitt diskutiert werden.

Das maschinelle Lernen ist inhärent fehleranfällig. Egal wie sorgfältig Modelle trainiert werden, besteht niemals Garantie dafür, dass ihre Vorhersagen in der Praxis stets korrekt sind. Zum einen sollte ML deshalb in keinem Fall verwendet werden, in welchem ein konkret beschreibbarer Algorithmus zur Lösung eines Problems bekannt ist. Konkrete Algorithmen sind deutlich zuverlässiger, da deren Korrektheit und Fehlerfreiheit überprüft werden kann. Zudem benötigen sie keine Trainingsdaten, sind für Menschen nachvollziehbar und können ohne Fachwissen im Bereich des ML implementiert werden. Zum anderen sollte maschinelles Lernen insbesondere dann nicht eingesetzt werden, wenn nuancierte Beschränkungen eingehalten werden müssen, wie etwa betriebliche Vereinbarungen oder rechtliche Anforderungen [Hum22, S. 12-13].

Eine weitere Einschränkung von ML und KI im Allgemeinen ist, dass diese nicht tatsächlich intelligent sind. Maschinelles Lernen ist in der Lage, komplexe Muster und Funktionen in großen Datenmengen zu approximieren, doch echtes Lernen findet nicht statt. Das ML lernt nicht, sondern imitiert lediglich die Trainingsdaten mithilfe angewandter Statistik. Ein tatsächliches Verständnis des Problems besteht nicht. Aus diesem Grund steht das Wort „erlernt“, in diesem Kapitel stets in Anführungszeichen, wenn es um maschinelles Lernen und künstliche Intelligenz geht.

Ebenfalls eine hochrelevante Einschränkung des maschinellen Lernens ist der sogenannte Bias (wörtlich ins Deutsche übersetzt: Voreingenommenheit). Grob gesagt beschreibt der Bias die Tendenz eines Modells, bestimmte Muster zu bevorzugen. Es gibt viele Quellen davon im ML.

Dabei ist Bias in den Daten, welche zum Training verwendet werden, besonders problematisch, da der Erfolg von maschinellem Lernen stark von der Qualität und Ausgewogenheit der Trainingsdaten abhängt. Bias kann etwa dadurch entstehen, dass im Web nur ein kleiner Teil der Nutzer einen Großteil des Contents generiert. Oder auch durch die Diskriminierung von Minderheiten — etwa Afroamerikaner im US-amerikanischen Justizsystem — welche sich letztendlich durch mehr Festnahmen von Personen dieser Ethnie auch

in den Polizeidatenbanken widerspiegelt. So könnte ein Modell lernen, dass Afroamerikaner häufiger Verbrechen begehen, obwohl dies nicht der Fall ist. Auch die Auswahl der Daten, welche zum Training verwendet werden, kann zu Bias führen. So basieren beispielweise viele medizinische Anwendungen bis zu 80% auf Daten aus westlichen, gebildeten, industrialisierten, wohlhabenden und demokratisierten Ländern[2] — obwohl diese nur 12% der Weltbevölkerung ausmachen. Ebenfalls zu berücksichtigen ist, dass es sich bei Daten inhärent um historische Aufzeichnungen handelt, welche nicht zwangsweise die aktuellen oder zukünftigen Umstände widerspiegeln [Hum22, S. 48-49].

Eine weitere Quelle von Bias sind die ML-Anwendungen selbst. Algorithmischer Bias kann während dem Training auftreten. Dieser ist die einzige Form von Bias im maschinellen Lernen, welcher anhand bestimmter Metriken leicht quantifiziert und mittels bestimmter Anpassungen effektiv verringert werden kann. Problematischer ist hingegen der Bias in der Interaktion mit den Anwendungen selbst. Zu Bias führen kann die Art und Weise, wie diese Anwendungen ihre Vorhersagen darstellen und wie nachvollziehbar dabei der Einfluss bestimmter Faktoren für die Nutzer ist. Nicht-Experten nehmen oft an, KI sei im Allgemeinen intelligent und ziehen deshalb unter Umständen den Fehlschluss, diese Kompetenz würde sich auch auf andere Bereiche übertragen, wenn gleich das Modell in diesen gar nicht trainiert wurde. Auch kann Bias mehr Bias erzeugen. Dieses Phänomen wird auch als *second order bias* bezeichnet und hat das Potenzial, zu einem Teufelskreis zu führen. Manche ML-Anwendung verwenden das Feedback der Nutzer zur Verbesserung des Modells, sodass voreingenommene Interaktionen mit der Anwendung den Bias nur weiter verstärken [Hum22, S. 49-50].

Über die hier präsentierten fachlichen Einschränkungen hinaus, darf der CO_2-Fußabdruck des maschinellen Lernens nicht außer Acht gelassen werden. Datenintensive Anwendungen wie ML benötigen enorme Rechenkapazitäten und verbrauchen dadurch große Mengen an Energie. Das Forschungsfeld *Green AI* beschäftigt sich bereits mit der Reduktion negativer Effekte von KI auf den Klimawandel, doch müssen sich Forschende, Entwickelnde und Nutzende von ML-Anwendungen dieser Problematik bewusst sein und mit entsprechender Sorgfalt handeln [Hum22, S. 7].

Alles in allem ist Bewusstsein für diese Limitationen des maschinellen Lernens wichtig. Bewusstsein über die inhärente Fehleranfälligkeit ermöglicht es, Vorkehrungen zur Fehlerbehandlung und -toleranz zu treffen. Bewusstsein darüber, dass KI keine tatsächliche Intelligenz besitzt, hilft bei der Einordnung der Kompetenz von ML-Anwendungen. Obwohl es unmöglich ist, Bias vollständig zu vermeiden, hilft das Bewusstsein über diesen bei einer sorgfältigen Auswahl der Trainingsdaten und bei der kritischen Reflexion über die Anwendungen. Dabei besteht eine Analogie zu menschlicher Intelligenz: Auch diese hat Grenzen, ist fehleranfällig, nicht perfekt und kann durch Bias beeinflusst werden.

2 Englisch: Western Educated Industrialized Rich Democratic (WEIRD) countries

2.4 GROSSE SPRACHMODELLE (LLMS)

Das Language Modeling (LM) ist ein Forschungsfeld der künstlichen Intelligenz, welches sich mit der Generierung von Sprache befasst. In diesem Kontext wird auch von generativer KI gesprochen, wobei dieser Begriff sich nicht nur auf Sprachmodelle beschränkt. Dabei werden Sprachmodelle verwendet, um die Wahrscheinlichkeiten von aufeinander folgenden Worten in einem gegebenen Kontext abzubilden. Anhand dieser Modelle kann sowohl Text vervollständigt – sozusagen weitergeschrieben — werden, als auch fehlende Wörter in einem Satz ergänzt werden [Zha+23, S. 1]. Ein Sprachmodell könnte beispielsweise zurate gezogen werden, um in folgendem Satz[3] die leere Stelle mit einem passenden Wort zu füllen: *Ich aß mit einer* _____ *zu Abend*. Das Sprachmodell könnte hier etwa die Worte *Gabel* oder *Freundin* vorschlagen, jeweils unter Angabe einer Wahrscheinlichkeit, wie gut der Vorschlag in diesen Kontext passt.

Große Sprachmodelle (LLMs) sind eine relativ junge Weiterentwicklung auf dem Gebiet des LM. Moderne (aber nicht notwendigerweise *große*) Sprachmodelle sind spezialisierte tiefe künstliche neurale Netze (ANNs), die üblicherweise viele sogenannte Parameter (Gewichte an den Neuronen innerhalb des Netzes) besitzen und auf enormen Datenmengen trainiert wurden. Diese Datenmengen schließen weite Teile des frei zugänglichen Internets, eine Vielzahl von Büchern und weitere Quellen ein. Der Begriff *großer* Sprachmodelle ist nicht formal definiert, grenzt sich aber von „normalen" Sprachmodellen durch die Verwendung der sogenannten Transformer-Architektur ab. Gleichzeitig haben LLMs typischerweise viele hunderte Milliarden Parameter [Zha+23, S. 3]. Die Transformer-Architektur und die zugrundeliegenden Mechanismen werden in Unterabschnitt 2.4.2 vorgestellt.

2.4.1 *Entstehung*

Erste Implementierungen von Sprachmodellen reichen bis in die 1990er Jahre zurück. Statistische Sprachmodelle (Statistical Language Models (STMs)) berechneten damals die Wahrscheinlichkeit für ein Wort, in Abhängigkeit von den vorherigen n Wörtern. Dieser Ansatz — auch n-Gramm-Modell — war ein wichtiger Meilenstein, litt jedoch unter dem exponentiell wachsenden Rechenaufwand mit steigendem n. Das Aufkommen neuraler Sprachmodelle (Neural Language Models (NLMs)) in den frühen 2000er Jahren hatte einen großen Einfluss auf die Entwicklung von zunehmend kompetenteren Sprachmodellen und die NLP als Ganzes [Zha+23, S. 1].

Ende der 2010er Jahre nahm die Forschung durch das Aufkommen von vortrainierten Sprachmodellen (Pre-trained Language Models (PLMs)) wieder an Fahrt auf. Dabei wurde ein Sprachmodell zunächst auf einem umfangreichen Datensatz vortrainiert und anschließend für eine bestimmte Aufgabe fein-angepasst [Zha+23, S. 2]. In diesem Kontext stellten Vaswani et al. 2017 in [Vas+17] erstmals die Transformer-Architektur vor, welche im Gegen-

3 Beispiel übernommen aus [Cho20, S. 605ff].

satz zu anderen Sprachmodell-Architekturen vollständig auf dem sogenannten *Attention*-Mechanismus basiert. Beide sind Kernelemente von LLMs und werden in Unterabschnitt 2.4.2 vorgestellt. Bahnbrechend war, dass Attention-basierte Transformer-Modelle bei geringeren Trainingskosten deutlich bessere Ergebnisse erzielten als bisher dagewesene Sprachmodelle [Vas+17, S. 9]. Zwei Jahre später, 2019, stellten Radford et al. in [Rad+19] erstmals fest, dass Transformer-Modelle dazu in der Lage sind, Aufgaben aus verschiedensten Domänen gut zu lösen, obwohl sie für diese Aufgaben nie explizit trainiert wurden, insofern diese auf einem sehr umfangreichen und diversen Trainingsdatensatz trainiert wurden [Rad+19, S. 10].

Eine weitere Realisierung über die Kompetenzen von größer werdenden Sprachmodellen stellten Brown et al. 2020 in [Bro+20] vor. Nicht nur werden diese Modelle mit steigender Größe insgesamt kompetenter, sie können darüber hinaus durch sogenannte *Few-Shot Prompts* zur Laufzeit und ohne weiteres Training auf bestimmte Aufgaben fein-angepasst werden. Prompts sind textuelle Eingaben, für welche das Sprachmodell Ausgaben generiert. Bei Few-Shot Prompts werden den Modellen Beispiel für die zu lösende Aufgabe präsentiert: etwa ein englischer Satz und die dazugehörige französische Übersetzung in einem bestimmten sprachlichen Stil, welcher bei der Übersetzung weiterer Sätze, für welche keine Beispiele vorliegen, berücksichtigt werden soll. Die Anpassung auf bestimmte Aufgaben ohne die Notwendigkeit eines erneuten Trainings eliminiert einen signifikanten Teil des Trainingsaufwands und ermöglicht einen weitaus flexibleren Einsatz der Modelle. In einigen Fällen erreicht die Kompetenz mittels Few-Shot Prompts sogar beinahe die Leistung von explizit für die Aufgabe trainierten Modellen [Bro+20, S. 3-9].

Trotz der zunehmenden Kompetenz größer werdender Sprachmodelle, waren diese jedoch nicht automatisch in der Lage, Anweisungen von Nutzern sinnvoll zu befolgen. Ausgaben der Modelle könnten inkorrekt, schlicht nicht hilfreich, oder sogar toxisch sein. Eine Lösung zur Verbesserung des sogenannten *Alignment* (in Deutsch als Orientierung oder Ausrichtung übersetzbar) wurde 2022 von Ouyang et al. in [Ouy+22] vorgestellt: Fine-Tuning mit menschlichem Feedback. Dabei wird das Verhalten des Modells in eine Richtung angepasst, welche von Menschen als wünschenswert identifiziert wurde und so das Alignment verbessert [Ouy+22, S. 5].

Die Firma OpenAI verwendete diese Erkenntnisse, um das Alignment ihres Modells GPT-3 zu verbessern [Ope22]. Aus dieser Entwicklung entstand schlussendlich der Chatbot ChatGPT, eines der beiden Werkzeuge, deren Einsatz in dieser Arbeit näher untersucht wird.

2.4.2 *Attention und die Transformer-Architektur*

Die Transformer-Architektur basiert auf dem Attention-Mechanismus. Beide werden in diesem Unterabschnitt oberflächlich beleuchtet. Die Idee hinter dem Mechanismus ist es, die Relevanz von Wörtern im Kontext anderer Wörter, durch ein spezialisiertes künstliches neuronales Netz abzubilden. Ein solches ANN bildet einen zentralen Baustein des Gesamtmodells und ermöglicht es,

die Abhängigkeiten zwischen Worten im Rahmen des Trainings zu „erlernen“. Mithilfe dessen wird nicht nur eine zentrale Herausforderung der NLP — die Identifikation der relevantesten Worte in einem gegebenen Kontext — lösbar. Ein Sprachmodell, in welchem dieser Baustein mehrfach eingesetzt ist, verfügt darüber hinaus über viele Kompetenzen auf dem Gebiet der Verarbeitung natürlicher Sprache. Die konkreten Gründe dafür sind aufgrund der Blackbox-Eigenschaften des ML nahezu unmöglich nachzuvollziehen. Doch entsteht ein sehr tiefes und hochkompetentes Modell, wird dieser Baustein mit herkömmlichen ANNs und weiter spezialisierten Mechanismen wie *Multi-Head Attention* und *Self-Attention* kombiniert [Vas+17, S. 3-9].

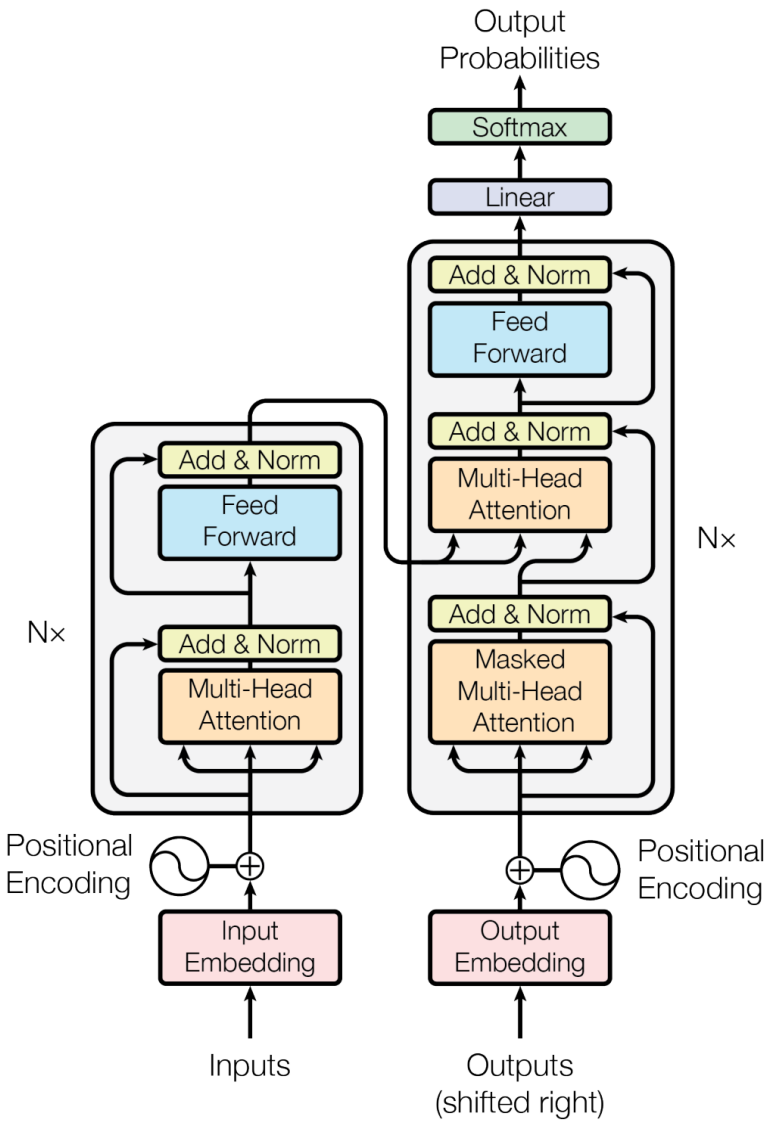


Abbildung 2.6: Architekturdiagramm des Transformer Modells. Die Abbildung ist übernommen aus [Vas+17, Figure 1].

Abbildung 2.6 zeigt ein Diagramm der Transformer-Architektur. Die links abgebildete Komponente ist der sogenannte *Encoder*, die rechts abgebildete

Komponente der *Decoder*. Die Aufgabe des Encoders ist es, eine Eingabesequenz in eine abstrakte interne Repräsentation dieser zu überführen, welche mit relevanten Informationen angereichert ist. Der Decoder nimmt die Repräsentation der Eingabe entgegen und generiert schrittweise einzelne Ausgaben. Die Ausgaben des Decoders werden darüber hinaus als zusätzliche Eingabe für diesen verwendet. Die im Diagramm grau hinterlegten Boxen stellen Komponenten dar, die mehrfach (N-mal) seriell konkateniert werden und so sehr tiefe Encoder und Decoder ergeben. In beiden Komponenten tritt der Attention-Mechanismus mehrfach auf [Vas+17, S. 2-4].

2.4.3 *Kompetenzen und Limitationen großer Sprachmodelle*

Große Sprachmodelle entwickeln sich aktuell rasch weiter. Die in diesem Abschnitt vorgestellten Kompetenzen und Limitationen sind daher nur eine Momentaufnahme, welche sich auf Quellen stützt, die den Stand bis zum Frühjahr und Sommer 2023 abbilden. Darüber hinaus sind Aussagen bezüglich Fähigkeiten und Kompetenzen von LLMs mit Vorsicht zu genießen und bewusst offen formuliert. Inwiefern davon gesprochen werden kann, dass diese Modelle tatsächlich bestimmte Fähigkeiten besitzen, ist gegenwärtig unmöglich zu bewerten. Generiert ein LLM eine Ausgabe, die eine korrekte logische Schlussfolgerung beinhaltet, so ist dies nicht notwendigerweise ein Beweis für die Fähigkeit zum logischen Denken — die Ausgabe wirkt lediglich so, als besitze das Modell diese Fähigkeit. Dennoch wird im Folgenden von Fähigkeiten und Kompetenzen gesprochen, um die Leistungen von LLMs zu beschreiben, ohne dabei sperrige Umschreibungen zu verwenden, um diesen Umstand zu beschreiben. Es wird dabei jedoch keine Aussage über die tatsächliche Intelligenz der Modelle getroffen.

Emergent Abilities

Während die Kompetenzen herkömmlicher Sprachmodelle vor allem die Generierung von Sprache umfassen, zeigen große Sprachmodelle eine überraschende Anzahl an weiteren Kompetenzen: sogenannte *Emergent Abilities*. Diese Fähigkeiten heißen „emergent“ (im Sinne von „auftauchend“), weil sie in kleineren Sprachmodellen nicht zutage treten, sondern nur in großen Sprachmodellen festgestellt werden konnten, ohne dass diese explizit darauf trainiert wurden. Ebenfalls trägt zu dem Namen bei, dass die Gründe für das Auftreten dieser Fähigkeiten bis heute nicht vollständig verstanden sind. Drei dieser Emergent Abilities sind das *In-Context Learning*, das *Instruction Following* und das *Step-by-Step Reasoning* [Zha+23, S. 2-4]:

- In-Context Learning bezeichnet die Fähigkeit, sich aus dem Kontext des Prompts (der Eingabe) heraus weitere Fähigkeiten anzueignen, ohne dass das Modell explizit dafür trainiert wurde. Etwa kann die Übersetzung in einem bestimmten Stil, aus Beispielen für diesen Stil gelernt werde, wie im Beispiel von Few-Shot Prompts (siehe Unterabschnitt 2.4.1). Ein weiteres Beispiel sind einfache arithmetische Fähigkeiten, wie etwa

die Addition bzw. Subtraktion dreistelliger Zahlen nach dem Vorbild von Beispielen.

- Nicht nur Beispiele können verwendet werden, um LLMs zu instruieren. Große Sprachmodelle besitzen die als Instruction Following bezeichnete Fähigkeit, Anweisungen in natürlicher Sprache gewissermaßen zu „verstehen“ und zu befolgen. Auf diese Weise können sie ebenfalls Aufgaben lösen, für welche sie nie explizit trainiert wurden.

- Die Kompetenz von LLMs wird durch ihre Fähigkeit zum Step-by-Step Reasoning weiter gesteigert. Werden Modelle im Prompt dazu aufgefordert, eine Aufgabe schrittweise zu lösen, folgen sie dieser Aufforderung und geben in ihrer Ausgabe vor einem finalen Ergebnis auch Zwischenschritte an. Diese eine Prompting-Strategie ist als Chain of Thought (CoT) bekannt. Gerade im Kontext komplexer Aufgaben (etwa mathematische Textaufgaben) können große Sprachmodelle auf diese Weise eine gute Lösung erzielen, während kleinere Modelle hier üblicherweise scheitern. Dieses Verhalten weist Parallelen zum logischen Denken und dem Ziehen von Schlussfolgerungen auf — daher der englische Name „Reasoning“. Wie in Unterabschnitt 2.3.4 besprochen verfügt KI zwar nicht über tatsächliche Intelligenz und replizieren lediglich Muster aus ihren Trainingsdaten, doch ist die Kompetenzsteigerung durch diese Fähigkeit nicht von der Hand zu weisen.

Zwar ist der Grad, zu welchem die Modelle die oben aufgeführten Fähigkeiten tatsächliche besitzen schwer zu quantifizieren, doch erlauben die Emergent Abilities es LLMs, zahlreiche komplexe Aufgaben zu lösen und in vielen Anwendungsgebieten eingesetzt zu werden. Die folgende Aufzählung deren erhebt keinen Anspruch auf Vollständigkeit. Große Sprachmodelle können Sprache verstehen, Text in andere Sprachen übersetzen, Texte vervollständigen und Fragen beantworten (sowohl auf Basis ihres im Training erworbenen, umfangreichen Weltwissens, als auch auf Basis von Quellen, die zur Laufzeit zur Verfügung gestellt werden). Sie können auch Quellcode generieren, Schlussfolgerungen im Alltagskontext und im mathematischen Kontexts generieren und sogar Prüfungsaufgaben lösen, welche ursprünglich zur Bewertung der Lernleistungen von Menschen entworfen wurden. In einigen Bereichen reicht die Leistung bestimmter Modelle dabei an die von Menschen heran. So etwa die des Modells GPT-4 in den Bereichen Coding, Mathematik oder Computer Vision. Neben diesen Fähigkeiten sind LLMs sehr robust gegenüber Rauschen, Störungen und Fehlern. Trotz Tippfehlern in den Instruktionen oder Unregelmäßigkeiten in den Beispielen sind sie in der Lage dazu, Prompts sinnvoll zu bearbeiten. Aufgrund dieser Robustheit sind LLMs hervorragend für den Einsatz in der Praxis geeignet [Zha+23, S. 29-35].

Fehleranfälligkeit und Halluzinationen

Jedoch besitzen große Sprachmodelle auch signifikante Limitationen. Erneut besteht in der folgenden Aufzählung kein Anspruch auf Vollständigkeit. Im

Allgemeinen erzielen LLMs keine guten Ergebnisse bei Aufgaben, welche weitreichende Planung oder gedankliche Sprünge erfordern. Ihre Ausgaben sind teilweise instabil und variieren stark in Abhängigkeit von Prompts und können sich sogar faktisch widersprechen. Auch können ihre Ausgaben sogenannte Halluzinationen enthalten: generierte, inkorrekte Informationen, welche im Konflikt mit gegebenen Quellen stehen oder nicht durch externe Quellen verifiziert werden können und quasi „frei erfunden“ wirken. So können Halluzinationen in den Ausgaben großer Sprachmodelle zu Missinformation führen. Abbildung 2.7 zeigt ein Beispiel einer Halluzination. Darüber hinaus beschränkt sich das Weltwissen von LLMs auf deren Trainingsdaten, sodass die Modelle an Aufgaben scheitern, deren Bearbeitung neue Informationen erfordert. Diese Limitation ist bei einigen Werkzeugen in Teilen bereits durch einen Internetzugang gelöst, jedoch in einigen Bereichen immer noch ein Hindernis. Eine weitere Limitation ist die Anwendung von Mathematik und die Durchführung nummerischer Berechnungen. Trotz ihrer Fähigkeit zum Generieren sinnvoller Schlussfolgerungen sind die Leistungen von LLMs in diesem Bereich sehr inkonsistent [Zha+23, S. 29-35].

Abbildung 2.7: Beispiel für eine Halluzination, bei der inkorrekte Informationen gegneriert wurden. Zwar bezeichnet die Abkürzung LLMs korrekterweise große Sprachmodelle, doch steht die Abkürzung im Kontext des ML für Reinforcement Learning Human Feedback (RLHF). Grafik übernommen und übersetzt aus [Zha+23, Fig. 8 (b)]

Bias

Bias ist eine weitere signifikante Limitation, die insbesondere große Sprachmodelle betrifft. Die zum Training verwendeten Daten sind tendenziell nicht repräsentativ für die Gesellschaft, sondern können homogene Weltanschauungen, Vorurteile oder Stereotype widerspiegeln. Aus dem Internet extrahierte Daten machen einen Großteil der Trainingsdaten von LLMs aus. Dabei sind Phänomene wie Toxizität, Hassrede, gesellschaftspolitischer Bias, Sexismus, Dehumanisierung und weitere niemals restlos auszuschließen. Die Modelle tendieren dazu, den Bias der Trainingsdaten zu reproduzieren und verstärken — das gilt ebenfalls für die oben genannten Phänomene. Gemeinsam mit dem Menschen innenwohnenden Bias und der menschlichen Tendenz echtes Verständnis in Ausgaben von LLMs hineinzuinterpretieren, birgt der Einsatz großer Sprachmodelle ein hohes Schadenspotential. Es besteht das Potenzi-

al für die Verstärkung von Bias und einer homogenen Weltanschauung, das Potenzial vorsätzlichen Missbrauchs und der Verbreitung von Fehlinformation und Hass. Aufklärung und Bewusstsein über diese Risiken sind daher essenziell für den verantwortungsvollen Umgang mit großen Sprachmodellen [Ben+21, S. 616-619]. Darüber hinaus ist der sogenannte *Confirmation Bias* zu beachten. Menschen neigen zur Bevorzugung von Informationen, welche ihre eigene Meinung bestätigen [Pet22]. In der Erwartung, die Ausgaben der Werkzeuge sein korrekt, entsteht so leicht die Neigung, Fehler zu übersehen und zu fehlerhafte Ausgaben zu akzeptieren.

Prompt Engineering

Die Leistung von LLMs wird stark von den verwendeten Prompts (den Eingaben) beeinflusst. Je nachdem, wie die Prompts formuliert sind, können die Modelle sehr unterschiedliche Ausgaben generieren. Die Entwicklung von Strategien und Prompts, um Aufgaben möglichst gut durch LLMs lösen zu lassen wird auch als Prompt Engineering bezeichnet. Die in Unterabschnitt 2.4.3 vorgestellte CoT Prompting-Strategie scheint beispielsweise zu besseren Ausgaben führen zu können, während bei der Aufnahme weiterer Hinweise oder zu vermeidende Themen in die Prompts ein gegenteiliger Effekt auftreten kann [Zha+23, S. 21]. Prompt Engineering kann auch dazu beitragen, Bias in den Ausgaben zu minimieren [Yan+23, S. 17]. Das Maß, in welchem Prompt Engineering die Kompetenzen großer Sprachmodelle beeinflusst, ist letztendlich jedoch schwer zu beziffern.

Token Limit

Eine weitere Einschränkung in der Interaktion mit großen Sprachmodellen ist das sogenannte *Token Limit*. Dabei werden Eingaben vor der Bearbeitung durch LLMs in sogenannte Tokens übersetzt — ihre Ausgaben machen die Modelle in Form von Tokens, welche wiederum in Text übersetzt werden. LLMs und damit auch Werkzeuge, welche diese verwenden, sind je Anfrage auf eine bestimmte Anzahl dieser Tokens beschränkt. Ein- und Ausgabe zählen dabei gegen dieselbe Beschränkung. Die Eingabe und deren zugehörige Ausgabe können gemeinsam nie größer als das Token Limit sein [Ope23d].

Illustriert wird diese Einschränkung hier anhand von ChatGPT. Die Details der Berechnung von Tokens unterscheiden sich zwischen verschiedenen Modellen und Werkzeugen — bei diesem Werkzeug entspricht ein Token etwa vier Buchstaben [Ope23d] (in der englischen Sprache). Abbildung 2.8 zeigt ein Beispiel für die Tokenisierung von Text und Quellcode. Dieses Beispiel illustriert ebenfalls, dass Quellcode tendenziell mehr Tokens als Text entspricht, was insbesondere beim Einsatz im Software Engineering relevant sein kann. Der Text im Beispiel umfasst 131 Buchstaben, welche in 45 Tokens übersetzt werden. Der Code entspricht 366 Buchstaben, welche in 149 Tokens übersetzt werden. Damit umfasst der tokenisierte Code circa 19% mehr Tokens pro Buchstabe als der tokenisierte Text.

Abbildung 2.8: Ein Screenshot des *Tokenizers* [Ope23c] von OpenAI. Der oben eingegebene Text wird in Tokens übersetzt und unten bunt illustriert. Dabei entsprechen benachbarte Buchstaben in derselben Farbe jeweils einem Token.

Im praktischen Umgang mit dem Chatbot ist das Token Limit ausschlaggebend für den Kontext, welchen ChatGPT je Anfrage berücksichtigen kann. OpenAIs genaue Implementierung des Token Limits ist nicht öffentlich einsehbar, doch scheint es etwa wie folgt zu funktionieren: Übersteigt der Chatverlauf das Token Limit, so wird der älteste Teil des Kontexts abgeschnitten. Das funktioniert ähnlich zu dem Sliding Window Prinzip: Werden bei erreichtem Token Limit 150 neue Tokens für einen weiteren Prompt und dessen Ausgabe benötigt, so werden die ältesten 150 Tokens abgeschnitten. Der Chatbot „vergisst“ gewissermaßen den Anfang des Chatverlaufes.

Volatile Kompetenzen LLM-basierter Werkzeuge

Diese Einschränkung betrifft nicht große Sprachmodelle direkt, sondern Werkzeuge, welche LLMs verwenden. Die vielfältigen und schwer quantifizierbaren Kompetenzen von LLMs treten bei der Verwendung von Werkzeugen noch undeutlicher hervor. Untersuchungen zeigen, dass die Leistung LLM-basierter Werkzeuge im Verlauf weniger Monate stark variieren kann. Wie und zu welchem Zeitpunkt dabei die verwendeten Modelle im Hintergrund aktualisiert, angepasst oder weiterentwickelt werden, ist für Nutzer nicht transparent. So kann das scheinbar selbe Werkzeuge zu unterschiedlichen Zeitpunkten sehr inkonsistente Ausgaben liefern. Eine Untersuchung am Beispiel von ChatGPT zeigt diese Unterschiede deutlich [CZZ23]: Hierbei wurden beiden Model-

le eine Reihe von Fragen aus verschiedenen Disziplinen unter Auswahl der Modelle GPT-3.5 und GPT-4 als Eingabe übergeben. Das geschah einmal im März 2023 und einmal im Juni 2023. In einigen Disziplinen änderten sich die Leistungen eines Modells dramatisch. So beantwortete ChatGPT unter Verwendung von GPT-4 im März 84% der Fragen aus dem Bereich *Math I* korrekt, während es im Juni nur noch 51,1% waren. Im selben Zeitraum stieg die Leistung von GPT-3.5 von 49,6% auf 76,2%, obwohl laut den Aussagen des von OpenAI — der Firma hinter dem Chatbot — GPT-4 ein kompetenteres Modell sei, welches grundsätzlich kompetenter, wenn auch langsamer als GPT-3.5 sei [Ope23a]. Die Leistung beider Modelle unterliegt auch in anderen Bereichen — etwa der Beantwortung politischer Fragestellungen oder der Generierung von Code — großen Schwankungen, sowohl aufwärts als auch abwärts. In anderen Bereichen blieb ihre Leistung relativ konsistent. Insgesamt hat sich die Leistung der Modelle über alle Disziplinen nicht durchweg verschlechtert. Mehr scheinen sich die Kompetenzen der Modelle zwischen einigen Disziplinen verschoben zu haben. Diese Veränderung könnte darauf zurückzuführen sein, dass die Verbesserung der Modelle für einzelne Aufgaben ungeahnte Effekte auf deren Leistung in anderen Bereichen haben kann — selbst für die Entwickler der Modelle [CZZ23]. Ändert sich die Modelle hinter den Werkzeugen, ändern sich zudem unter Umständen auch Aspekte der Limitationen, Fähigkeiten oder der Effektivität von Prompting-Strategien. Im Kontext des Einsatzes von LLM-basierten Werkzeugen ist Bewusstsein über diese Inkonsistenz höchst relevant.

Verarbeitungsgeschwindigkeit von Informationen

Bei der Verarbeitung von Text bringen große Sprachmodelle einen entscheidenden Vorteil: Geschwindigkeit. Zwar können die Ausgaben der Modelle inkorrekt sein, Bias beinhalten und auf historische Trainingsdaten limitiert sein, doch umfasst das Weltwissen von LLMs ein gewaltiges Volumen an Informationen, zusammengesetzt aus weiten Teilen des frei zugänglichen Internets, einer Vielzahl von Büchern und einigen weiteren Quellen ein. Die Modelle können dabei helfen, diese Informationen zugänglicher zu machen — ähnlich zu herkömmlichen Websuchmaschinen. Das Maß dieser Unterstützung illustriert durch eine Gegenüberstellung der menschlichen Lesegeschwindigkeit und der Geschwindigkeit von ChatGPT deutlich: In der englischen Sprache lesen Menschen im Durchschnitt etwa 238 Wörter pro Minute [Bry19]. Im Mittel bestehen Wörter der englischen Sprache aus 4,79 Buchstaben [Nor12], sodass Menschen etwa 1140 Buchstaben pro Minute lesen. Wie oben erwähnt, geschieht die Ein- und Ausgabe von LLMs anhand sogenannter Tokens, welche je Anfrage auf ein Token Limit beschränkt sind. Ein Token entspricht für ChatGPT etwa 4 Buchstaben. Umgerechnet auf Tokens pro Minute ergibt sich für Menschen eine Geschwindigkeit von $1140/4 = 285$ Tokens pro Minute. Die Variante des Modells GPT-3.5 mit kleinster Kontextlänge[4] besitzt selbige von 4097 Tokens [Ope23b]. Die Generierung einer Ausgabe dieses Mo-

4 Stand September 2023

dells dauert nur wenige Sekunden. In diesem Beispiel sei von konservativen 5 Sekunden ausgegangen. Verarbeitet das Werkzeug also innerhalb von 5 Sekunden 4097 Tokens, so ergibt sich eine Verarbeitungsgeschwindigkeit von $4097 \cdot 12 = 49164$ Tokens pro Minute, was um den Faktor $49164/1140 \approx 43,1$ schneller ist, als die Lesegeschwindigkeit von Menschen. Selbst unter konservativen Annahmen und ausgehend von der GPT-3.5-Variante mit kleinster Kontextlänge, illustriert dieses Beispiel einen großen Vorteil der Fähigkeit von LLMs, Fragen zu beantworten.

Problematische Erhebung der Trainingsdaten

Ein durchaus problematischer Aspekt der Trainingsdaten ist die Art und Weise, wie diese gesammelt wurden. Diese Daten bestehen üblicherweise aus weiten Teilen des frei zugänglichen Internets, einer Vielzahl von Büchern und einigen weiteren Quellen — und diese Daten wurden in der Regel ohne die Zustimmung der Dateneigentümer zum Training der Modelle verwendet. So wurde von einer Reihe an Autoren Klage gegen OpenAI eingereicht, dem Unternehmen hinter ChatGPT mit dem Vorwurf, das Unternehmen habe urheberrechtlich geschützte Inhalte aus Büchern ohne Zustimmung der Autoren verwendet [Dav23]. Auch gegen die Verwendung frei zugänglicher Inhalten aus dem Internet wurde geklagt [Mau23]. Problematisch sei in diesem Zusammenhang die Verwendung von personenbezogenen Daten, welche in den Trainingsdaten enthalten sein können. Eine abschließende Klärung der Rechtslage steht zwar noch aus[5], jedoch könnte die Verwendung von Trainingsdaten ohne Zustimmung der Dateneigentümer in Zukunft ein Hindernis für die Verwendung von LLMs darstellen.

Einordnende Bemerkung

Alles in allem fällt eine eindeutige und verlässliche Evaluation der Kompetenzen und Limitationen von LLMs schwer. Es ist unklar über welche Fähigkeiten LLMs in welchem Maße verfügen. Das liegt unter anderem an der hohen Komplexität der Modelle und der Blackbox-Natur von ML im Allgemeinen, an der Vielzahl an Parametern, die Einfluss auf die Kompetenz der Modelle haben, als auch daran, dass es sich um ein aktives Forschungsfeld handelt.

Bei der Evaluation der Fähigkeiten großer Sprachmodelle tut sich darüber hinaus ein philosophischer Aspekt auf: Wo beginnen Verständnis und Intelligenz? Wie oben erwähnt, verwendet diese Arbeit bewusst offene Formulierung bezüglich der Kompetenzen von LLMs, da gegenwärtig unmöglich zu bewerten ist, inwiefern tatsächlich davon gesprochen werden kann, dass die Modelle bestimmte Fähigkeiten besitzen. Generiert ein LLM eine Ausgabe, die eine korrekte logische Schlussfolgerung beinhaltet, so ist dies nicht notwendigerweise ein Beweis für die Fähigkeit zum logischen Denken — die Ausgabe wirkt lediglich so, als besitze das Modell diese Fähigkeit. Kann in diesem Kontext von einem Verständnis der Aufgabe gesprochen werden? Ist es denkbar,

5 Stand: September 2023

dass die menschliche Intelligenz das emergente Phänomen eines leistungsstarken Gehirns ist? Diese Diskussion verlässt den Rahmen dieser Thesis sowie die Grenzen des menschlichen Verständnisses und wird deshalb nicht eingehender besprochen.

2.5 STAND DER FORSCHUNG VON KI-GESTÜTZEM SE

Der Einsatz von KI im Software Engineering ist ein aktives und rasch wachsendes Forschungsfeld. Dabei ist das maschinelle Lernen der populärste Ansatz und wird im größten Teil der bis 2021 veröffentlichten Publikationen verwendet. Unter diesen Publikationen beschäftigt sich zudem fast die Hälfte mit dem Testen von Software. Auch die Wartung und die Entwicklung von Software sowie die Anforderungsanalyse wurden in der Vergangenheit bereits untersucht, während andere Bereiche des SE bislang wenig Aufmerksamkeit erhielten [CCC23, S. 1-5].

Insbesondere die in dieser Arbeit untersuchte Unterstützung des Software Engineering durch den Einsatz LLM-basierter Werkzeuge ist ein relativ junges Forschungsfeld, an welchem gegenwärtig besonders intensiv geforscht wird. Allein in den ersten neuneinhalb Monaten des Jahres 2023 erschienen tausende Publikationen[6], welche den Mehrwert des Einsatzes einzelner Werkzeuge in einzelnen Domänen des Software Engineering untersuchten. Eine umfassende Untersuchung und Vorstellung dieser Literatur verlässt den Rahmen dieser Arbeit und wäre im Umfang selbst eine eigenständige Arbeit. Auch aufgrund der gegenwärtigen Dynamik des Feldes wäre eine solche Übersicht binnen weniger Wochen veraltet. Aus diesem Grund werden im Folgenden nur einige ausgesuchte Beispiele aufgeführt, ohne Anspruch auf Vollständigkeit:

- Ziegler et al. evaluierten in [Zie+22] im Juni 2022 den Einfluss des Werkzeuges GitHub Copilot auf die Produktivität von Softwareentwicklern. Eine große Zahl an Probanden bewertete ihre eigene Produktivität im Kontext der Unterstützung durch den Assistenten. Parallel wurde eine Korrelation zwischen Aspekten des Nutzungsverhaltens der Probanden und deren wahrgenommene Produktivität festgestellt. Die Ergebnisse lassen auf großes Potenzial für einen Effizienzgewinn schließen.

- Ahmad et al. stellten im Februar 2023 in [Ahm+23] ihre Untersuchungsergebnisse zum Einsatz von ChatGPT als Unterstützung in der Softwarearchitektur vor. Sie kommen zu dem Ergebnis, dass großes Potenzial für den gewinnbringenden Einsatz des Chatbots in diesem Feld besteht, und suchen dies in zukünftiger Arbeit praktisch zu evaluieren. Die Ergebnisse deuten auch darauf hin, dass KI im Software Engineering nicht nur durch Automatisierung unterstützten kann, sondern auch durch Kollaboration mit menschlichen Experten.

6 16.300 Suchergebnisse zu „Large Language Models in Software Engineering“ auf Google Scholar bei betrachtetem Zeitraum 01.01.2023 bis 10.10.23 [Goo].

- Ross et al. präsentierten im März 2023 in [Ros+23] den Prototypen eines LLM-basierten Systems, welches als eine Art Programmierassistent das Verstehen, Schreiben und Arbeiten mit von Code unterstützt. Auf Basis von Untersuchungen mit einigen Probanden kommen sie zu einem sehr positiven Ergebnis: Das System steigert die selbst wahrgenommene Produktivität der Probanden und fiel insgesamt durch vielfältige Unterstützungsmöglichkeiten auf.

- Im April 2023 evaluierten Yetistieren et al. in [Yet+23] die Qualität des generierten Codes mehrerer LLM-basierter Werkzeuge. Hierbei sollten die Werkzeuge Code zu Aufgaben aus einem Benchmarking-Datensatz generieren. Während der Code nicht selten technische Schulden (Code Smells, etc.) beinhaltete, war er in den meisten Fällen syntaktische korrekt. Die Werkzeuge lieferten besonders dann gute Ergebnisse, wenn die Aufgabe klar beschrieben war.

- Zhang et al. stellten im Juli 2023 in [Zha+] ihre Untersuchungen zum Einsatz von ChatGPT als Unterstützung im Abrufen von Informationen aus einem Anforderungsdokument vor. Ihre Evaluation lässt auf Potenzial für und einen Effizienzgewinn durch die Unterstützung des Chatbots auf diesem Gebiet schließen, wenn auch der Rahmen der Evaluation im Umfang begrenzt war.

- Ebenfalls zum Thema Requirements Engineering präsentierten im Juli 2023 Ray et al. in [TR+23] ihre Untersuchungen bezüglich der Übersetzung von menschenlesbaren Anforderungen in ein standardisiertes maschinenlesbares Format mittels großer Sprachmodelle. Zwei für diese Aufgabe fein-angepasste Modelle produzierten zwar keine fehlerfreien Ergebnisse, könnten jedoch die Arbeitsgeschwindigkeit bei der Standardisierung vieler Anforderungen erhöhen und gerade für unerfahrenere Entwickler hilfreich sein.

- Dell'Acqua et al. stellten in [Del+23] im September 2023 eine Untersuchung des Einsatzes von ChatGPT im Kontext komplexer wissensintensiver Aufgaben im Bereich der Unternehmensberatung vor. Zwar findet diese Untersuchung außerhalb des SE statt, doch haben die Ergebnisse trotzdem Relevanz für das SE. Die Leistungen einer großen Zahl von Probanden wurden praktisch evaluiert — sowohl mit Unterstützung durch den Chatbot als auch ohne — und die Ergebnisse gegenübergestellt. Diesen zufolge erhöhte die Unterstützung von ChatGPT die Leistung der Probanden bei der Wissensarbeit signifikant. Demnach habe der Einsatz von generativer KI transformatives Potenzial für die Art und Weise, wie Menschen arbeiten. Es solle nicht die Frage gestellt werden, ob KI in den Workflow von Wissensarbeitern integriert wird, sondern wie dies geschieht. Auch das Software Engineering ist Wissensarbeit, sodass vermutet werden kann, dass der beschriebene positive Effekt sich auch dort bemerkbar macht.

Die vorliegende Arbeit verfolgt ein ähnliches Ziel wie die oben aufgeführten Beispiele: Eine Untersuchung welchen Einfluss der Einsatz von LLM-basierten Werkzeugen auf das Software Engineering hat. Dabei unterscheidet sich diese Arbeit in zwei Punkten von den oben aufgeführten Beispielen: im Umfang und in der Methodik. Zum einen betrachtet diese Arbeit den gesamten SE-Prozess — nicht nur die Softwarearchitektur, das Requirements Engineering oder die Generierung von Code. Zum anderen verfolgt diese Arbeit einen empirischen Ansatz, bei welchem konkrete Erfahrungen von Experten im praktischen Umgang mit den Werkzeugen im Fokus stehen, welche aus der Praxis des Software Engineering gewonnen sind. Nach bestem Wissen und Gewissen existieren bis dato keine Publikationen, welche den Einsatz LLM-basierter Werkzeuge in diesem Umfang und mit dieser Methodik untersuchen.

Unter den oben aufgeführten Beispielen sind jene von Ross et al. und Ziegler et al. die am nächsten verwandten Arbeiten. Ross et al. erprobten ihren Prototyp ebenfalls empirisch mit Probanden aus der Praxis des Software Engineering und sammelten deren Erfahrungen ebenfalls mittels mehrerer Fragebögen. Der zentrale Unterschied besteht im Umfang, da die Forscher sich nur auf das Schreiben von Code konzentrierten. Auch Ziegler et al. untersuchten den Einfluss der Unterstützung von GitHub Copilot auf empirischer Basis und erfassten die Wahrnehmung der Probanden. Mit der Untersuchung von Korrelationen zwischen Nutzungsverhalten und wahrgenommener Produktivität gingen sie dabei einen Schritt weiter.

3 POTENZIALANALYSE

Der Einsatz LLM-basierter Werkzeuge birgt großes Potenzial, das Software Engineering gewinnbringend zu unterstützen, wie ein Blick in die Literatur zeigt (siehe Abschnitt 2.5). Dieses Kapitel präsentiert eine breitgefächerte Potenzialanalyse, welche den Mehrwert der Unterstützung dieser Werkzeuge im SE auf dem Stand Mai 2023 aus verschiedenen Perspektiven betrachtet. Ziel ist die Identifikation von Potenzialen — wobei ein Potenzial die Kombination eines konkreten Einsatzgebietes mit einem konkreten Werkzeug darstellt, in welchem das Werkzeug bereits heute effektiv eingesetzt werden kann, um den Software Engineering Prozess zu unterstützen. Die Ergebnisse dieses Kapitels dienen als Grundlage für die Pilotuntersuchungen, welche in Kapitel 4 vorgestellt werden. Dort liegt der Fokus auf den Werkzeugen und Einsatzgebiete, welche hier als besonders vielversprechend identifiziert wurden.

In diesem Kapitel werden zunächst in Abschnitt 3.1 die grundlegende Methodik und die verwendeten Ansätze beschrieben. Anschließend stellen Abschnitt 3.2 und Abschnitt 3.3 die verwendeten Ansätze im Detail dar. Zuletzt werden in Abschnitt 3.4 die Ergebnisse zum Fokus der Pilotuntersuchungen zusammengeführt.

3.1 METHODIK

Zur Identifizierung der Potenziale werden zwei Ansätze betrachtet, welche sich jeweils auf einen der beiden Aspekte eines Potenzials fokussiert. Ansatz A betrachtet den Prozess des Software Engineering als solchen, und identifiziert darin Domänen und Aufgabenbereiche, in welchen die Kompetenzen großer Sprachmodelle gewinnbringend eingesetzt werden können. Ansatz B betrachtet eine große Bandbreite an derzeit verfügbaren KI-Werkzeugen und evaluiert deren Eignung für den Einsatz im Software Engineering sowie deren Eignung für die Erprobung in den Pilotuntersuchungen. Die Ergebnisse beider Ansätze werden in Abschnitt 3.4 zu einer Grundlage für die Pilotuntersuchungen zusammengeführt.

Eine Definition von gewinnbringendem und effektivem Einsatz ist jedoch nicht trivial. Generell ist die Messung von Produktivität im Software Engineering eine Herausforderung [Kal22]. Ebenso die Identifizierung der Faktoren, die zum Erfolg oder Misserfolg im SE beitragen, der zudem nur langfristig beurteilt werden kann [TPK20]. Die vorliegende Arbeit beschränkt sich daher auf eine erste Orientierung. Im Kontext dessen wurden die folgenden drei Aspekte betrachtet:

- Einfluss auf den wirtschaftlichen Erfolg: Obwohl ein Anstieg in Qualität oder Quantität der entwickelten Software schwer zu quantifizieren

ist und häufig zeitversetzt auftritt, stellt eine Steigerung hier einen signifikanten Mehrwert dar. Im Kern geht es im SE darum, mit einem bestimmten Budget ein bestimmtes Ziel in vorgegebener Zeit zu erreichen. Insofern KI-Werkzeuge dabei unterstützen, ein Ziel besser oder schneller zu erreichen, bedeutete dies einen großen Einfluss auf die eigene Wettbewerbsfähigkeit.

- Einfluss auf die Zufriedenheit: Eine erhöhte Zufriedenheit auf Seite der Kunden oder Engineers ist ein weiterer Faktor. Dieser könnte sich etwa in der Übernahme von langweiligen oder repetitiven Tätigkeiten durch KI äußern. Obwohl sich der Einfluss von Zufriedenheit auf den wirtschaftlichen Erfolg schwer messen lässt, ist das Wohlbefinden der Beteiligten ein signifikanter Mehrwert, der sich zudem direkt abfragen und so quantifizieren lässt. Ein impliziter marktwirtschaftlicher Vorteil zufriedener Mitarbeitenden ist außerdem, dass diese dadurch eher geneigt sind, auch langfristig für das Unternehmen zu arbeiten.

- Einfluss auf Kompetenzen und Fähigkeiten der Engineers: Auch ein positiver Einfluss auf die Fähigkeiten der Engineers stellt einen signifikanten Mehrwert dar. Unterstützt ein Werkzeug die Ausübung eine Tätigkeit oder hilft dabei, neue Fähigkeiten zu erlernen, kann dies als gewinnbringend betrachtet werden. Unterstützung kann hier etwa heißen, dass die Tätigkeit effektiver, schneller oder mit geringerer Hürde ausgeführt oder erlernt werden kann.

Diese Aspekte dienen als Indikatoren für einen gewinnbringenden, effektiven Einsatz von KI-Werkzeugen im Software Engineering.

3.2 ANSATZ A — POTENZIALE IN DOMÄNEN DES SE

Im Kontext dieser Evaluation ist ein Potenzial die Kombination aus einem konkreten Einsatzgebiet und einem konkreten Werkzeug, welches dort effektiv eingesetzt werden kann. Ansatz A befasst sich der ersten dieser Hälften: den Einsatzgebieten im SE. Das Software Engineering ist ein komplexer Prozess, der viele Aufgabenbereiche und Tätigkeiten beinhaltet. Um in diesem Prozess systematisch Potenziale für den Einsatz von KI-Werkzeugen identifizieren zu können, bedarf es einem umfassenden Überblick. Zu diesem Zweck dient die Darstellung der *BeST Foundation* als Orientierung. Die *BeST Foundation* ist eine von Accso ins Leben gerufene Initiative, deren Ziel die Vermittlung der Kerninhalte des Software Engineerings ist. Das Akronym *BeST* steht für „**Be**schleunigte **S**oftware**t**echnik“. Die Foundation definiert den Begriff folgendermaßen:

> „Modernes Software Engineering beinhaltet einen riesigen Fundus an verschiedenen Practices, d.h. Anwendungen von Methoden, Techniken und Werkzeugen. **Beschleunigte** Softwaretechnik bedeutet, diese Practices zu kennen, situationsgerecht auszuwählen und ge-

> zielt anzuwenden, mit dem Ziel, die **Effizienz** und die **Effektivität** des Projekts zu optimieren um damit in der **industriellen Praxis** besonders **produktiv** und somit **wettbewerbsfähig** agieren zu können“ [Sol23].

Die Foundation unterteilt das Software Engineering systematisch in sieben Domänen, welche konkrete Aufgabenbereiche beinhalten. Eine Übersicht über die Inhalte der Foundation ist in Abbildung 3.1 abgebildet. Nach der Foundation ist das SE ein iterativer Prozess, dem die Domäne *1. Vorgehensstrategie* vorweg steht. Eine Iteration beginnt in der Domäne *2. Requirements & Analyse*, gefolgt von *3. Architektur & Design* usw., und endet in der Domäne *6. Infrastruktur & Betrieb*. Parallel zu den Iterationen stellt die Domäne *7. Projektmanagement* sicher, dass die Projektziele bestmöglich erreicht werden. Ansatz A orientiert sich an der Darstellung der Foundation und identifizieren Potenziale für die jeweiligen Domänen und Aufgabenbereiche. Der nachfolgende Unterabschnitt beschreibt die sieben Domänen im Detail.

3.2.1 *Domänen des SE nach BeST*

Die Domäne *1. Vorgehensstrategie* steht vor dem iterativen Kernprozess. Sie befasst sich mit der Planung des Prozesses und der Entscheidung über Strategien und Methodik auf mehreren Ebenen — alles in Abhängigkeit zu den individuellen Anforderungen des Projekts. Mit der Domäne *2. Requirements & Analyse* beginnt der Kernprozess. Hierbei versuchen die beteiligten Personen ein gemeinsames Verständnis des zu lösenden Problems zu erzielen, um die bestmögliche IT-Lösung für die Endanwender entwickeln zu können. Anschließend geht es in *3. Architektur & Design* um die Entwicklung oder Modifikation einer Softwarearchitektur zur Lösung des Problems. Die Architektur ist die grundlegende Organisation eines Systems und spiegelt sich in den Komponenten und deren Beziehungen zueinander wider. Eine gute Architektur verfolgt Ziele, die sie zu erreichen hilft und verkörpert die Grundsätze für die Gestaltung eines Systems. Die Domäne *4. Implementierung* dreht sich um die konkrete Umsetzung von einer IT-Lösung in Form von Code. Wichtig ist dabei nicht nur funktionierende, sondern auch gut gefertigte Software zu entwickeln und die Qualität dieser abzusichern. Die Absicherung der Softwarequalität führt direkt in die nächste Domäne: *5. Test*. In dieser geht es darum, eine geeignete Teststrategie festzulegen und automatisierte Tests zu schreiben, um die Qualität der IT-Lösung weiter abzusichern. Die letzte Domäne des iterativen Kernprozesses ist die Domäne *6. Infrastruktur & Betrieb*. Sie befasst sich mit dem dauerhaften Betrieb und der Unterstützung einer entwickelten IT-Lösung. Ein Kernaspekt ist dabei die zugrundeliegende Infrastruktur zu pflegen und Prozesse zu automatisieren. Die Domäne *7. Projektmanagement* begleitet das SE über den gesamten Prozess. Es geht darum sicherzustellen, dass die Projektziele bestmöglich erreicht werden und ein optimaler Trade-Off zwischen diesen Zielen erreicht wird.

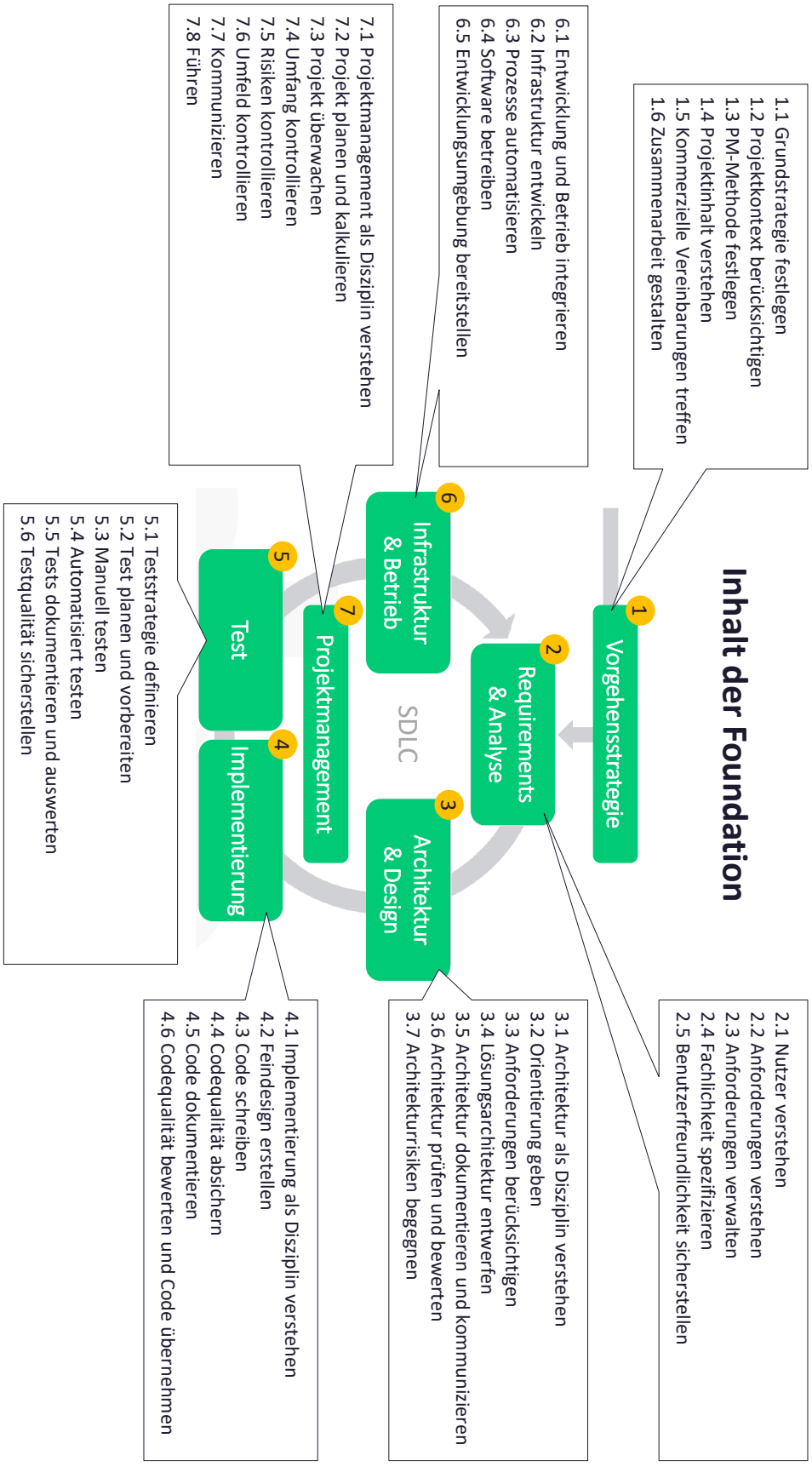


Abbildung 3.1: Übersicht über die Domänen und Aufgabenbereiche im Softwareentwicklungsprozess nach der *BeST Foundation*. Grafik entnommen aus [Sol23].

3.2.2 *Potenziale nach Domänen*

Dieser Unterabschnitt geht die Domänen und Aufgabenbereiche des Software Engineering durch und diskutiert mögliche Potenziale für den Einsatz von KI-Werkzeugen, auf Basis der in Unterabschnitt 2.4.3 besprochenen Eigenschaften von LLMs in Kombination der in Abschnitt 3.1 beschriebenen Aspekte für gewinnbringenden Einsatz im SE. Die hier vorgestellen Potenziale sind hypothetische Überlegungen und kommen am ehesten einem Brainstorming gleich.

In der Domäne *1. Vorgehensstrategie* scheinen keine konkreten Potenziale für die Unterstützung durch LLM-basierte Werkzeuge zu bestehen. Zwar verfügen große Sprachmodelle über vielseitige Kompetenzen und ein umfassendes Weltwissen und könnten etwa als ergänzende Wissensressource eingesetzt werden, doch ist diese Domäne sehr abstrakt und situationsabhängig. Diese Distanz zur Praxis gestaltet die Identifizierung konkreter Potenziale herausfordernd und macht es denkbar, dass die Werkzeuge hier keine systematische Anwendung finden.

Die Domäne *2. Requirements & Analyse* liegt näher an der Praxis, sodass die Identifizierung von Potenzialen hier leichter fällt. Es ist denkbar, dass große Sprachmodelle im Aufgabenbereich *2.2 Anforderungen verstehen* unterstützen können, indem sie mittels ihres umfangreichen Weltwissens den Zugang zu Domänenwissen als zusätzliche Wissensressource erleichtern. Auch könnten sie Fragen zu großen Anforderungsdokumenten beantworten und diese zusammenfassen oder aber mithilfe geschickter Prompts dazu genutzt werden, Gespräche mit Personas simulieren, in welchen das Sprachmodell als eine Persona imitiert. Personas sind fiktive Charakterprofile, die typisch Nutzer eines Systems darstellen. Sie kommen unter anderem in der Softwareentwicklung zum Einsatz, um die Nutzer und deren Bedürfnisse besser verstehen und berücksichtigen zu können [Nie13, S. 2]. Insbesondere dort ist jedoch Vorsicht geboten, da LLMs inhärent fehleranfällig sind und ihre Ausgaben Bias beinhalten. Im Aufgabenbereich *2.3 Anforderungen verwalten* könnten die Werkzeuge ebenfalls zur Unterstützung im Umgang mit Anforderungsdokumenten eingesetzt werden. In 2.4 Fachlichkeit spezifizieren wäre die Generierung von Anforderungsdokumenten oder fachlichen Spezifikationen denkbar und in *2.5 Benutzerfreundlichkeit sicherstellen* könnten Werkzeuge eine automatisierte und „intelligente“ Überprüfung der Benutzerfreundlichkeit ermöglichen. Insgesamt scheinen LLM-basiere Werkzeuge in der Domäne *2. Requirements & Analyse* das Potenzial zu haben, die Arbeit zu beschleunigen und potenziell auch repetitive Arbeiten abkürzen — etwa das Schreiben von Spezifikationen oder anderen großen Dokumenten. Ebenfalls möglich wäre sogar eine erhöhe Qualität der Arbeit, da mithilfe des Weltwissens mehr Faktoren berücksichtigt werden können.

In der Domäne *3. Architektur & Design* scheint großes Potenzial für den Einsatz von LLM-basierten Werkzeugen zu bestehen. Im Aufgabenbereich *3.3 Anforderungen berücksichtigen* können die Sprachmodelle als zusätzliche Wissensquelle für Implikationen von Anforderungen auf die Architektur dienen, wobei auch dort aufgrund von Fehleranfälligkeit und Bias Vorsicht geboten ist.

Darüber hinaus könnten LLMs im Aufgabenbereich *3.4 Lösungsarchitektur entwerfen* dabei helfen, Architekturentwürfe zu generieren oder Referenzarchitekturen vorzuschlagen, auf Basis des Weltwissens. Auch denkbar wäre die Visualisierung von Architektur durch die Generierung von Code zum Erstellen von Architekturdiagrammen. Das würde sowohl diesen als auch den Aufgabenbereich *3.5 Architektur dokumentieren und kommunizieren* unterstützen. Ähnlich wie bei der Überprüfung der Benutzerfreundlichkeit könnte in *3.6 Architektur prüfen und bewerten* eine automatisierte und „intelligente" Überprüfung der Architektur auf ihre Qualität möglich sein. Zuletzt könnten im Aufgabenbereich *3.7 Architekturrisiken begegnen* Handlungsoptionen zur Reaktion auf Risiken vorgeschlagen werden. Ähnlich wie in der zweiten Domäne besteht in dieser das Potenzial, die Arbeit zu beschleunigen und potenziell auch ein besseres Ergebnis zu erzielen.

In der umsetzungsnächsten Domäne, *4. Implementierung*, scheinen die meisten Potenziale zu liegen. So könnten LLM-basierte Werkzeuge etwa im Aufgabenbereich *4.2 Feindesign erstellen* geeignete Design Patterns oder Entwürfe vorschlagen. In *4.3 Code schreiben* könnte die Fähigkeit, Code zu generieren nützlich sein, um etwa Boilerplate Code, simple Funktionen oder Code in Domain-Specific Languages (DSLs) zu generieren. Auch ist Unterstützung im Debugging sowie Unterstützung durch Code-Vervollständigung denkbar. Im Aufgabenbereich *4.4 Codequalität absichern* könnten die Werkzeuge bei der automatisierten und „intelligenten" Überprüfung der Codequalität unterstützen oder inkonsistente Code-Styles sowie potenzielle Sicherheitslücken identifizieren. In *4.5 Code dokumentieren* könnten die Werkzeuge etwa im kleinen Rahmen einzelne Codekommentare generiert werden oder im großen Rahmen sogar Wiki-ähnliche Quellcode-Dokumentation generieren, welche im Nachhinein nur noch von den Engineers überprüft werden muss. Im Aufgabenbereich *4.6 Codequalität bewerten und Code übernehmen* könnten die Werkzeuge das Code-Review und den Kompatibilitätscheck zu bestehendem Code unterstützen. Elementar dabei ist, ob das Weltwissen der Sprachmodelle für diese spezifische Aufgabe ausreicht. Insgesamt scheinen LLM-basierte Werkzeuge in der Domäne *4. Implementierung* das Potenzial zu haben, die Arbeit an vielen Stellen zu beschleunigen und potenziell auch repetitive Arbeiten abkürzen — etwa das Schreiben von Code oder Dokumentation.

Potenziale scheinen auch in der Domäne *5. Test* zu bestehen. So könnten die Fähigkeit LLMs, Schlussfolgerungen zu generieren im Aufgabengebiet *5.3 Manuell testen* unterstützen, indem die Werkzeuge etwa zu testende Szenarien identifizieren oder Testdatensätze für diese Szenarien generieren. Auch an dieser Stelle ist der in den Modellen enthaltene Bias zu beachten, der die generierten Daten in Richtung von Klischees und Stereotypen lenken könnte. Die generierten Testfälle wären auch im Aufgabenbereich *5.4 Automatisiert testen* relevant. Darüber hinaus könnten die Werkzeuge dort Code für automatisierte Tests generieren, ähnlich wie sie Code für die Implementierung generieren. Auch die Aufgabengebiete *5.5 Tests dokumentieren und auswerten* und *5.6 Testqualität sicherstellen* könnten von den Werkzeugen auf ähnliche Weise unterstützt werden, wie es bei 4.5 und 4.6 im vorherigen Absatz dargestellt

ist. Alles in allem scheint die Unterstützung durch LLMs-basierte Werkzeuge in dieser Domäne ähnliches Potenzial zu haben, wie in der Implementierung, sodass auch hier die Arbeit an vielen Stellen beschleunigt werden könnte.

Die Domäne *6. Infrastruktur & Betrieb* ist wiederum weniger nah an der Implementierung, sodass die Identifizierung von Potenzialen hier erneut herausfordernder ist. Im Aufgabengebiet *6.3 Prozesse automatisieren* könnten die Werkzeuge potenziell bei der Generierung von Skripts zur Automatisierung von Infrastruktur und Prozessen unterstützen — Stichwort Infrastructure as Code (IaC). Unter Umständen ist die Fähigkeit von LLMs, Code zu generieren auch auf Kommandozeilenbefehle zur Automatisierung von Prozessen anwendbar. So könnten die Werkzeuge auch hier als Wissensressource dienen. Ähnlich sieht es bei *6.4 Software betreiben* aus, wo die Werkzeuge als Wissensressource bei der Anwendung von Versionsupdates oder Patches genutzt werden können. Insgesamt scheint in dieser Domäne jedoch weniger konkretes Potenzial zu bestehen, als in den vorherigen.

Die letzte Domäne, *7. Projektmanagement*, scheint erneut zu situationsabhängig zu sein, um eindeutig konkrete Potenziale identifizieren zu können. Zwar können große Sprachmodelle auch hier als Wissensressource eingesetzt werden, doch ist der konkrete Nutzen davon schwer zu fassen. So ist denkbar, dass keine Projektmanagerin aktuell einen signifikanten Unterschied durch die Unterstützung von LLMs in dieser Domäne feststellen würde.

3.3 ANSATZ B — VERFÜGBARE WERKZEUGE

Die erste Hälfte eines Potenzials ist ein konkretes Aufgabengebiet. Die zweite Hälfte ein konkretes Werkzeug, welches in dort gewinnbringend eingesetzt werden kann. Dieser Abschnitt stellt eine Reihe KI-basierter Werkzeuge vor, welche in Vorbereitung auf die Pilotuntersuchungen evaluiert wurden. Dabei werden die jeweiligen Werkzeuge vorgestellt und ihre Eignung für den Einsatz im SE generell diskutiert, als auch für die Erprobung im Rahmen der Pilotuntersuchungen. Bei der Eignung für den Einsatz im SE steht dabei der, in Abschnitt 3.1 beschriebene Mehrwert im Vordergrund. Für die Eignung zur Erprobung in den Pilotuntersuchungen sind folgende Kriterien relevant:

- Verfügbarkeit: Das Werkzeug muss im gesetzten Zeitrahmen von Ende April 2023 bis Ende Juli 2023 verwendbar sein. Relevant sind dabei ebenfalls die Verfügbarkeit in Deutschland zu diesem Zeitpunkt, der preisliche sowie rechtliche Rahmen für den Einsatz im kommerziellen Umfeld.

- Daten- und Geheimnisschutz: welche Daten bei Verwendung des Werkzeuges erhoben werden und welche Garantien der Betreiber bezüglich dem Schutz dieser Daten gibt. Dieses Kriterium ist höchst relevant für den Einsatz im kommerziellen Umfeld, da es sich bei dem Code, der Geschäftslogik von Anwendungen, etc. um schützenswerte Güter handelt. Dabei geht es nicht nur um Güter, die Accso selbst gehören, sondern auch um Güter ihrer Kunden. Die Verwendung von Drittanbieter-Diensten kann stets die Vertraulichkeit von Daten gefährden, was auch

für und sogar insbesondere für LLM-basierte Werkzeuge gilt. LLMs greifen unter Umständen sehr tief in die eigene Arbeit ein und sind dazu eine relativ junge Entwicklung, deren Einsatz teilweise eine offene bzw. ungeklärte Rechtslage beinhaltet. Inwiefern dabei der Schutz geistigen Eigentums gewährleistet ist, ist nicht immer klar.

- Unterstütze Programmiersprachen: Je mehr Programmiersprachen und Frameworks unterstützt werden, in desto mehr Projekten kann das Werkzeug potenziell eingesetzt werden.

Die Evaluation geschah über den Zeitraum von Anfang April bis Mai 2023 in Vorbereitung auf die Pilotuntersuchungen. Insofern sind die hier präsentierten Informationen lediglich eine Momentaufnahme. Evaluiert wurden insgesamt 16 Werkzeuge. Während zwar zu diesem Zeitpunkt Dutzende, wenn nicht Hunderte von KI-Werkzeugen im Web verfügbar sind, beschränkt sich die Evaluation auf einige wenige populäre und vielversprechende Kandidaten. Eine Analyse aller verfügbaren Werkzeuge wäre nicht praktikabel gewesen und hätte den Rahmen dieser Arbeit verlassen. Die evaluierten Werkzeuge wurden der Übersicht halber und entsprechend ihrer Art in vier Kategorien eingeteilt: „Intelligente“ Chatbots, KI-Pair-Programmer, verschiedene SE-Werkzeuge sowie Nicht-SE-Werkzeuge. Gegliedert nach diesen Kategorien sind im Nachfolgenden die evaluierten Werkzeuge aufgeführt.

3.3.1 *„Intelligente“ Chatbots*

„Intelligente“ Chatbots basieren auf großen Sprachmodellen und können auf Basis von Texteingaben, textuelle Ausgaben generieren. Sie verfügen über die, in Unterabschnitt 2.4.3 vorgestellten, Kompetenzen und Limitationen von LLMs und sind deshalb vielseitig einsetzbar. Neben der Generierung von Text oder Code können ihre Emergent Abilities bei vielen Aufgaben im SE unterstützen. Dabei sind die potenziellen Einsatzgebiete ähnlich zu denen, welche in Unterabschnitt 3.2.2 aus den Kompetenzen und Limitationen großer Sprachmodelle abgeleitet wurden: „Intelligente“ Chatbots können potenziell in der Anforderungsanalyse und Softwarearchitektur eingesetzt werden, die Implementierung und das Testen von Software unterstützen oder auch bei dem Betrieb von Software helfen und dort die Effizient sowie Qualität der Arbeit zu erhöhen. Das Wort Intelligenz in Namen dieser Kategorie steht in Anführungszeichen, da die Werkzeuge keine tatsächliche Intelligenz besitzen (siehe Unterabschnitt 2.3.4). Die folgenden „intelligenten“ Chatbots wurden evaluiert:

ChatGPT[1] ist ein von der Firma OpenAI entwickelter Chatbot, welcher auf den großen Sprachmodellen GPT-3.5 und GPT-4 basiert. Der Chatbot ist über den Browser verfügbar. Mehre Konversationen werden in verschiedenen Chats organisiert, die zu beliebiger Zeit pausiert oder fortgeführt werden können. Abbildung 3.2 zeigt einen Screenshot der Benutzeroberfläche. Zum Zeitpunkt

1 https://chat.openai.com (Zuletzt aufgerufen am 29.09.2023)

dieser Evaluation ist ChatGPT nach Registrierung eines Benutzerkontos für Privatpersonen kostenlos nutzbar, jedoch beschränkt auf Varianten des Modells GPT-3.5. Das kompetentere, wenn auch langsamere Modell, GPT-4, ist nur mit einer kostenpflichtigen Mitgliedschaft namens *ChatGPT Plus* nutzbar [Ope23a], welche derzeit 20 US-Dollar pro Monat kostet, und neben einem weiteren Modell auch die Nutzung von Plugins und weiteren Features ermöglicht. Eine auf kommerzielle Nutzung ausgelegte Lizenz gibt es zur Zeit der Untersuchung nicht[2]. Der Chatbot ist in Deutschland verfügbar unterstützt eine Vielzahl von Programmiersprachen sowie Frameworks, ohne dass diese explizit angegeben werden können. Bezüglich Datenschutz macht OpenAI keine Garantien — jeglicher Input kann gespeichert und für zukünftiges Training verwendet werden.

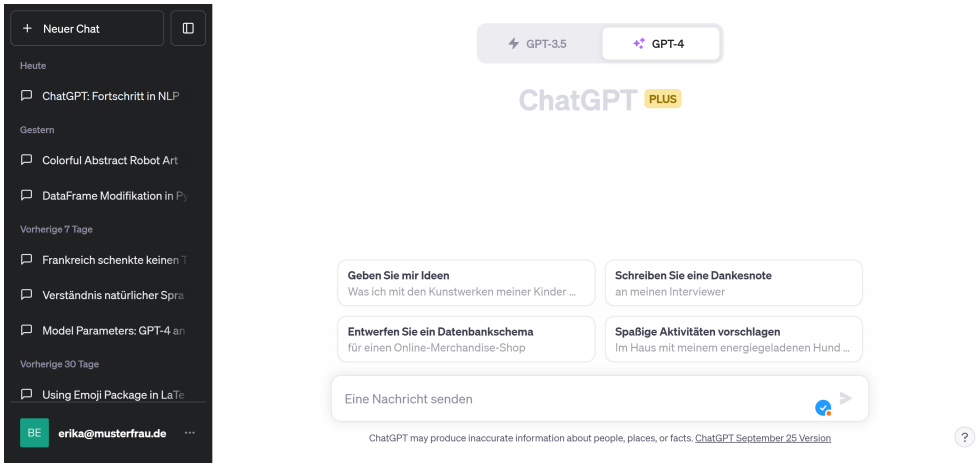


Abbildung 3.2: Ein Screenshot der Benutzeroberfläche von ChatGPT. Links ist der Chatverlauf zu sehen, mittig die Eingabemaske.

The New Bing[3] ist eine Erweiterung der von Microsoft entwickelten Suchmaschine, die Websuchen in natürlicher Sprache erlaubt und auf dem Sprachmodell GPT-4 basiert. Das Werkzeug kann gratis, jedoch ausschließlich in Browser Microsoft Edge verwendet werden. Konversationen sind auf fünf Antworten limitiert, insofern kein Microsoft Account verwendet wird. Das Werkzeug fasst die Ergebnisse von Suchen zusammen und referenziert auf die verwendeten Online-Quellen. Es scheint nicht auf den Einsatz im Software Engineering ausgelegt zu sein, doch kann The New Bing auch beim Coding unterstützen. Ähnlich wie bei ChatGPT bestehen keine Garantien bezüglich Datenschutz.

Google Bard[4] ist ein von Google entwickelter „intelligenter" Assistent, der jedoch nur beschränkte Coding-Unterstützung bietet und zum Zeitpunkt der Evaluation nicht in Deutschland verfügbar ist.

2 Ende August 2023 wurde *ChatGPT Enterprise* angekündigt: `https://openai.com/blog/introducing-chatgpt-enterprise` (Zuletzt aufgerufen am 29.09.2023)
3 `https://www.bing.com/new` (Zuletzt aufgerufen am 29.09.2023)
4 `https://bard.google.com` (Zuletzt aufgerufen am 29.09.2023)

LLaMA[5] ist ein von der Firma Meta AI entwickeltes großes Sprachmodell, welches 2023 in [Tou+23] vorgestellt wurde. Sein Name steht für „Large Language Model Meta AI“. Zwar ist LLaMA ein Sprachmodell und kein Chatbot, jedoch verschwimmen hier die Grenzen, da Chatbots im Kern Sprachmodelle verwenden, um ihre Antworten zu generieren. Zum Zeitpunkt der Untersuchung ist LLaMA nur auf Anfrage verfügbar.

Alpaca[6] (**LLaMA-based Language Model**) ist ein LLM, welches auf der Basis von Metas LLaMA entwickelt wurde. Dies geschah jedoch ohne Metas Zustimmung, sodass die Benutzung von Alpaca rechtlich kritisch ist und insbesondere in einem kommerziellen Umfeld nicht empfohlen werden kann. Ähnlich wie LLaMA selbst ist Alpaca ein Sprachmodell und kein Chatbot, kann jedoch ähnlich verwendet werden. Alpaca steht frei zum Download zur Verfügung und muss zur Verwendung lokal installiert und ausgeführt werden.

3.3.2 *KI-Pair-Programmer*

Bei KI-Pair-Programmern handelt es sich um direkt in die Entwicklungsumgebung integrierte Werkzeuge, die während der Arbeit einer Engineer textuelle Vorschläge an der Position des Cursors unterbreiten. Am ehesten ist die Unterstützung dieser Werkzeuge mit einer mächtigeren Auto-Vervollständigung zu vergleichen. Die Einbindung in die Integrated Development Environment (IDE) macht die Werkzeuge grundsätzlich einfach zu nutzen und ermöglicht es ihnen, kontextsensitive Vorschläge anzubieten, welche auf dem eigenen Code basieren. Durch ihre Einschränkung auf textuelle Vervollständigungen sind sie weniger vielseitig als Chatbots und primär auf die Unterstützung beim Schreiben von Code ausgelegt. Folgende Werkzeuge wurden in dieser Kategorie evaluiert:

GitHub Copilot[7] ist ein, von der Firma GitHub entwickelter Assistent, welcher auf dem Sprachmodell Codex basiert und bereits im Oktober 2021 öffentlich verfügbar war. Der Assistent steht als Plugin für gängige Entwicklungsumgebungen zur Verfügung, kann darin installiert und mithilfe eines GitHub Accounts verwendet werden. Die Nutzung kostet monatlich 10 US-Dollar für Privatpersonen und 19 pro Nutzer für Geschäftskunden. Der Assistent ist in Deutschland verfügbar und unterstützt eine Vielzahl gängiger Programmiersprachen: darunter C, C++, C#, Go, Java, JavaScript, PHP, Python, Ruby, Scala und TypeScript. Bezüglich des Datenschutzes wirbt GitHub mit „Industry Leading Privacy“ für Geschäftskunden. Die weiterentwickelte Version des Werkzeuges, Copilot X, ist derzeit nur für einen eingeschränkten Nutzerkreis verfügbar.

Bei **Amazon Code Whisperer**[8] handelt es sich um einen, von der Firma Amazon entwickelten Assistenten, der im April 2023 frei zugänglich wur-

5 https://ai.meta.com/blog/large-language-model-llama-meta-ai (Zuletzt aufgerufen am 29.09.2023)
6 https://github.com/cocktailpeanut/dalai (Zuletzt aufgerufen am 29.09.2023)
7 https://github.com/features/copilot (Zuletzt aufgerufen am 29.09.2023)
8 https://aws.amazon.com/codewhisperer (Zuletzt aufgerufen am 29.09.2023)

de. Ebenfalls kann der Assistent als Plugin für gängige Entwicklungsumgebungen installiert und dort mithilfe eines Accounts verwendet werden. Mit leicht beschränktem Funktionsumfang dürfen Privatpersonen das Werkzeug kostenlos verwenden, währen Geschäftskunden 19 US-Dollar pro Nutzer und Monat zahlen müssen. Ebenfalls ist das Werkzeug in Deutschland verfügbar und bietet für Geschäftskunden den Schutz des für die Vorschläge verwendeten Codes an. Am besten funktionieren die Programmiersprachen Java, Python, JavaScript, Typescript und C#, doch es werden auch weitere unterstützt.

tabnine[9] ist ein weiterer KI-Assistent, der sich an Software Entwickler richtet. Ebenfalls ist der Assistent als Plugin für gängige Entwicklungsumgebungen verfügbar und benötigt die Anmeldung eines Accounts. Ein stark limitierter Funktionsumfang steht gratis zur Verfügung. Für die volle Funktionalität können Teams von bis zu 100 Nutzern ein monatliches Abonnement 12 US-Dollar pro Nutzer abschließen. Über 100 Nutzer hinaus ist ein individuelles Angebot verfügbar. Datenschutz wird für alle Nutzer versprochen, unabhängig von der Art des Abonnements. Der Assistent ist in Deutschland verfügbar und unterstützt eine Vielzahl von Programmiersprachen und explizit auch einige Frameworks: Java, C++, C+, HTML, Css, JavaScript, TypeScript und PHP, sowie die Frameworks Angular, Node.js, React und weitere.

3.3.3 *Verschiedene SE-Werkzeuge*

Diese Kategorie umfasst ein breites Feld an Werkzeugen zum Einsatz im Software Engineering, welche nicht in die beiden vorherigen Kategorien passen. Sie können in weiterführenden SE-bezogenen Aufgabengebieten unterstützten, wie etwa in der Dokumentation, der Codeinspektion oder als spezialisierte Suchmaschinen. Die folgenden Werkzeuge fallen unter diese Kategorie:

Mintlify Writer[10] unterstützt beim Schreiben von „Inline“ Dokumentation — also Kommentaren im Quellcode. Auch dieses Werkzeug kann als Plugin in gängige Entwicklungsumgebungen installiert werden, kann dort jedoch auch ohne Anmeldung verwendet werden. Dazu muss der zu kommentierende Code markiert und ein Knopf gedrückt werden. Der generierte Kommentar wird dann über den markierten Codezeilen eingefügt. Konkrete Angaben zu den Kosten und dem Datenschutz gibt es nicht. Das Werkzeug ist aus Deutschland heraus nutzbar und unterstützt die Programmiersprachen JavaScript, Typescript, Java, Kotlin und Python.

Phind[11] ist ähnlich wie The New Bing eine Suchmaschine, welche Websuchen in natürlicher Sprache ermöglicht, ihre Ergebnisse zusammenfasst und auf Quellen verweist. Anderes als Bing ist Phind jedoch auf den Einsatz im Software Engineering spezialisiert. Neben der Websuche bietet Phind auch einige Funktionen an, die in Richtung eines „intelligenten“ Chatbots gehen — wie etwa die Identifizierung von Fehlern bei gegebenen Codeabschnitten. Das Werkzeug kann im Browser verwendet werden und ist kostenlos nutzbar. Es

9 https://www.tabnine.com (Zuletzt aufgerufen am 29.09.2023)
10 https://writer.mintlify.com (Zuletzt aufgerufen am 29.09.2023)
11 https://www.phind.com (Zuletzt aufgerufen am 29.09.2023)

ist von Deutschland aus nutzbar und unterstützt eine Vielzahl von Programmiersprachen und Frameworks, ohne dass diese explizit angegeben werden. Verlässliche Aussagen zum Datenschutz bestanden zur Zeit der Evaluation nicht.

Snyk[12] ist ein Werkzeug zur Codeinspektion, welches auf Sicherheitslücken und andere Probleme im Quellcode hinweisen kann. Welche Technologien das Werkzeug verwendet und ob große Sprachmodelle darunter sind, ist nicht öffentlich ersichtlich. Angeboten werden verschiedene Wege, das Werkzeug zu benutzen: etwa in der IDE, in der Cloud oder in weiteren Umgebungen. Eine kostenfreie Variante mit einer limitierten Zahl an Prüfungen steht zur Verfügung — die Kosten einer unlimitierten Variante hängt von den individuellen Anforderungen und genutzten Dienstleistungen ab. Unterstützt werden viele Programmiersprachen und Umgebungen, darunter JavaScript, Java, Python, PHP, .NET and C++, Docker, AWS, Red Hat und Jenkins. Das Werkzeug ist in Deutschland verfügbar. Die Situation bezüglich des Datenschutzes ist unklar.

OpenCommit[13] unterstützt bei der Formulierung von Commit-Nachrichten im Kontext der Versionsverwaltungssoftware Git. Das Werkzeug kann kostenfrei heruntergeladen und installiert werden und über die Kommandozeile, das Command Line Interface (CLI), verwendet werden, um Beschreibungen von in einem Commit enthaltenen Änderungen zu generieren. Das Werkzeug ist Open Source und verwendet intern die Application Programming Interface (API)-Dienste von ChatGPT, sodass ein ChatGPT Account zur Benutzung notwendig ist. Die Nutzung der API kostet Bruchstücke von Centbeträge je Anfrage und damit je generierter Commit-Nachricht. Es ist davon auszugehen, dass dieselben Programmiersprachen wie bei ChatGPT unterstützt werden und dieselbe Situation bezüglich des Datenschutzes besteht. Das Werkzeug ist in Deutschland verfügbar.

3.3.4 *Nicht-SE-Werkzeuge*

Werkzeuge aus dieser Kategorie haben keinen direkten Bezug auf das Software Engineering, haben jedoch trotzdem das Potenzial, Engineers in ihrem Arbeitsalltag zu unterstützen. Dabei geht es etwa um das Generieren von Meeting-Notizen, das Zusammenfassen von Dokumenten oder das Schreiben von E-Mails. Die folgenden Nicht-SE-Werkzeuge wurden evaluiert:

Luminous[14] ist ein großes Sprachmodell der Heidelberger Firma Aleph Alpha. Zum Zeitpunkt der Evaluation bietet es keine SE-bezogenen Funktionen an. Zum Funktionsumfang zählt das Zusammenfassen oder Verfassen von Texten, das Beschreiben von Bildern oder das Übersetzen zwischen einigen europäischen Sprachen. Ein Mehrwert für den Einsatz im Software Engineering könnte die Formulierung von E-Mails oder Blogbeiträgen sein, sodass die eingesparte Zeit für andere Aufgaben verwendet werden kann. Die Verwen-

12 https://snyk.io/de (Zuletzt aufgerufen am 29.09.2023)
13 https://github.com/di-sukharev/opencommit (Zuletzt aufgerufen am 29.09.2023)
14 https://www.aleph-alpha.com (Zuletzt aufgerufen am 29.09.2023)

dung des Werkzeuges wird je Anfrage abgerechnet, wobei die genauen Kosten von vielen Faktoren abhängen, sich im Mittel jedoch auf Bruchstücke von Centbeträge belaufen. Durch den Sitz in Heidelberg ist das Werkzeug „Made in Germany“, somit in Deutschland verfügbar und gibt hohe Standards für den Datenschutz an. Ein Vorteil ist, dass die Daten in Servern innerhalb der Europäischen Union verarbeitet werden, was die Einhaltung der DSGVO unter Umständen erleichtert. Auch besteht die Möglichkeit, das Werkzeug selbst zu hosten und so volle Kontrolle über die Daten zu behalten.

Otter.ai[15] ist ein KI-Werkzeug zur Transkription von Meetings. Nach der Einrichtung kann das Werkzeug zu Videoanrufen zugeschaltet werden, um die Gespräche zu transkribieren, zusammenzufassen und sogar visuell darzustellen. Diese Funktionalität birgt das Potenzial, die zahlreichen Meetings moderner Softwareentwicklung zu besser zu dokumentieren und so Zeit zu sparen. Verfügbarkeit, Kosten und Datenschutz wurden nicht tiefergehend untersucht.

Humata.ai[16] erlaubt es, PDF-Dateien hochzuladen und Fragen zu diesen in natürlicher Sprache zu stellen. Auch wenn das Software Engineering nicht das primäre Einsatzgebiet ist, kann das Werkzeug dort etwa die Arbeit mit umfangreichen Anforderungsdokumenten unterstützen oder bei Recherchetätigkeiten helfen. Humata.ai kann über den Browser verwendet werden. Verfügbarkeit, Kosten und Datenschutz wurden nicht tiefergehend untersucht.

Microsoft 365 Copilot[17] ist von Microsoft angekündigter Assistent, der die Arbeit mit Microsoft Office Anwendungen auf viele Arten unterstützen soll. Das Werkzeug soll bei der Formulierung von Texten oder E-Mails ähnlich wie GitHub Copilot helfen, es soll bei der Verwaltung von Kalender und E-Mail-Postfach unterstützen und an vielen weiteren Stellen die Arbeit erleichtern. Ähnlich wie die anderen Werkzeuge aus dieser Kategorie könnten Engineers durch diese Unterstützung mehr Zeit für andere Aufgaben haben. Zum Zeitpunkt der Evaluation ist Microsoft 365 Copilot noch nicht verfügbar.

3.4 FOKUS

Dieser Abschnitt führt die Ergebnisse der beiden Ansätze zusammen und legt den Fokus für die Pilotuntersuchungen fest. Nach Ansatz A scheinen die Werkzeuge vor allem in den umsetzungsnahen Domänen signifikante Unterstützung bieten zu können. Ansatz B zeigt, dass viele Werkzeuge potenziell geeignet sind, um den Software Engineering Prozess zu unterstützen. Beide Zwischenergebnisse sind hierbei eine Orientierung, aus welcher sich der Fokus der Pilotuntersuchungen ableitet und welche durch diese überprüft wird.

Zu erwarten ist, dass von den Domänen des Software Engineering vor allem in den Domänen *3. Architektur & Design*, *4. Implementierung* und *5. Test* ein Mehrwert durch die Unterstützung der Werkzeuge entsteht. Trotzdem wird

15 https://otter.ai (Zuletzt aufgerufen am 29.09.2023)
16 https://www.humata.ai (Zuletzt aufgerufen am 29.09.2023)
17 https://news.microsoft.com/de-de/wir-stellen-vor-microsoft-365-copilot (Zuletzt aufgerufen am 29.09.2023)

im Rahmen der Untersuchungen der gesamte Software Engineering Prozess betrachtet. Dabei werden die Untersuchungen so gestaltet, dass die in Abschnitt 3.1 aufgeführten Aspekte für effektiven Einsatz der Werkzeuge evaluiert werden: der Einfluss auf den wirtschaftlichen Erfolg, auf die Zufriedenheit sowie auf die Kompetenzen und Fähigkeiten der Engineers. Bezüglich der Werkzeuge werden die Untersuchungen durch den finiten Zeitrahmen und die Zahl der Probanden beschränkt. Um einen möglichst guten Überblick über den Einsatz LLM-basierter Werkzeuge im SE zu erhalten, ohne gleichzeitig zu tief in eine Richtung zu gehen, wird sich auf jeweils ein Werkzeug aus den relevantesten Kategorien beschränkt: ChatGPT als ein Vertreter „intelligenter“ Chatbots und GitHub Copilot als ein Vertreter der KI-Pair-Programmer. Beide Werkzeuge waren zum Evaluationszeitpunkt die ausgereiftesten ihrer jeweiligen Kategorien, schienen die höchste Qualität an Ausgaben zu liefern und waren daher die geeignetsten Repräsentanten für die Pilotuntersuchungen.

4 PILOTUNTERSUCHUNGEN

Der Fokus der Hauptuntersuchung liegt auf den Werkzeugen ChatGPT und GitHub Copilot, wie in Abschnitt 3.4 dargelegt. Es werden alle Domänen des Software Engineering betrachtet, jedoch besteht die Erwartung, dass die Werkzeuge vor allem in den Domänen *3. Architektur & Design*, *4. Implementierung* und *5. Test* unterstützen können. Entsprechend dieser Parameter wurde eine Gruppe Probanden zusammengestellt, welche die beiden Werkzeuge im Umfeld professioneller SE erproben sollten. Ziel war es, handfeste Erfahrung im Umgang mit den Werkzeugen zu sammeln und die Auswirkungen dieser auf das SE quantifizierbar zu machen.

Dieses Kapitel beschreibt zunächst in Abschnitt 4.1 den Versuchsaufbau und die Rahmenbedingungen der Untersuchung. Anschließend werden in Abschnitt 4.2 ein zentrales Element der Untersuchungen vor: den Fragebogen, mithilfe dessen die Erfahrungen der Probanden erfasst wurden. Dort werden sowohl die Fragen selbst, als auch die Antworten der Probanden vorgestellt. Abschnitt 4.3 vervollständigt die Antworten der Probanden um die von ihnen erprobten Anwendungsfälle und ordnet diese ein. Abschnitt 4.4 präsentiert und diskutiert Schlussfolgerungen, die aus den Ergebnissen der Untersuchung abgeleitet werden können.

4.1 VERSUCHSBESCHREIBUNG

Die Untersuchungen liefen von Ende April 2023 bis Ende Juli 2023. Eine Gruppe aus 19 Probanden erprobte in diesem Zeitraum die Werkzeuge ChatGPT und GitHub Copilot im Rahmen ihrer täglichen Arbeit im Software Engineering. Dabei kamen bei ChatGPT sowohl die kostenfreie Variante, als auch die kostenpflichtige Variante *ChatGPT Plus* und bei GitHub Copilot ausschließlich die *For Business*-Variante zum Einsatz. Obwohl Accso die Kosten für die Werkzeuge übernahm, entschieden sich einige Probanden für die kostenfreie Variante von ChatGPT. Unterabschnitt 3.3.1 und Unterabschnitt 3.3.2 geben einen Überblick über die Werkzeuge und deren verfügbare Varianten. Während Untersuchungszeitraums wurden wöchentliche Meetings abgehalten. Zweck dieser Meetings war es, weiteres Vorgehen zu planen, neue Informationen auszutauschen und sich gemeinsam mit den Probanden über neue Erfahrungen mit den KI-Werkzeugen auszutauschen. An diesen Meetings nahm ein großer Teil der Probanden regelmäßig teil. Am Ende dieses Zeitraums wurde mit jedem Probanden jeweils ein Interview durchgeführt, um dessen Erfahrungen festzuhalten.

Bei den Probanden handelt es sich um Software Engineers von Accso, die im Schnitt bereits über 7 Jahre im SE tätig waren. Diese Probanden verteilten sich auf die Standorte Darmstadt, Frankfurt, Köln, München, Kapstadt. Die

Auswahl erfolgte zum einen nach Interesse der Probanden, zum anderen in Abhängigkeit von deren Projektsituation, da die Werkzeuge nicht uneingeschränkt in jeglichen Projekten einsetzten werden konnten. Ein Hindernis für viele Projekte bestand in Bedenken von Accsos Kunden um Daten- und Geheimnisschutz (wie in Abschnitt 3.3 als Kriterium für den Einsatz der Werkzeuge erwähnt). Abhängig von der Projektsituation und persönlicher Präferenz erprobten die Probanden nach eigener Wahl eines der Werkzeuge oder beide Werkzeuge in beliebigen Domänen und Anwendungsfällen. Insgesamt haben 16 der Probanden ChatGPT und 11 GitHub Copilot getestet. Von den ChatGPT-Nutzern verwendeten 9 Probanden ausschließlich die kostenfreie Variante, 4 ausschließlich die kostenpflichtige Variante und 3 Probanden verwendeten beide Varianten. Teils war es den Probanden möglich diese Werkzeuge in Kunden- oder in internen Projekten einzusetzen, teils ausschließlich in privatem Kontext. Insgesamt haben 12 bzw. 13 ChatGPT in professionellem bzw. privatem Umfeld eingesetzt und 9 bzw. 3 GitHub Copilot in professionellem bzw. privatem Umfeld eingesetzt. Dabei verwendeten die Probanden ein breit gefächertes Spektrum an Programmiersprachen & Frameworks. Aufgrund dieser heterogenen Ausgangssituation unterschieden sich die Erprobungszeiträume der einzelnen Probanden und Werkzeugen — die Meisten hatten jedoch mehr als 3 Monate Erfahrung mit ChatGPT und 1 bis 3 Monate Erfahrung mit GitHub Copilot. Auf die genaue Verteilung wird in Abschnitt 4.2 tiefer eingegangen. Eine bestimmte Art der Einarbeitung mit den Werkzeugen wurde nicht vorgegeben. Es blieb den Probanden frei überlassen, inwiefern sie sich in die Werkzeuge einlesen wollten. In den wöchentlichen Meetings wurden regelmäßig Hinweise und Tipps gesammelt und geteilt. Eine Schulung oder explizites Onboarding hat nicht stattgefunden.

Innerhalb des letzten Monats des Untersuchungszeitraumes wurde je ein Interview pro Proband durchgeführt. Im Kern bestanden diese Interviews aus einem Fragebogen, welchen der Autor dieser Thesis gemeinsam mit dem Proband durchging. Interviews dauerten im Schnitt 60 Minuten (45 bis 75). Bis auf zwei Ausnahmen fanden alle Interviews via Video Call statt. Der Fragebogen wurde per Screenshare aufgelegt und vom Interviewer live bearbeitet, sodass die Probanden die Notizen sehen und korrigierend eingreifen konnten. Im Falle der zwei Ausnahmen fand das Interview in Präsenz statt. Dort wurde anstelle eines Screenshare auf denselben Bildschirm geschaut. Der Fragebogen lag in Deutsch und Englisch vor. Die Interviews wurden nach Präferenz des Probanden in einer der beiden Sprachen durchgeführt. Der Bogen wurde iterativ entwickelt. Insgesamt zwei Iterationen des Fragebogens wurden vorab mit zwei bzw. einem Probanden getestet und überarbeitet. Schlussendlich durchliefen alle Probanden (inklusive der Vorabtest-Kandidaten) die finale Version des Fragebogens. Diese ist in Abschnitt 4.2 beschrieben. Die englische Übersetzung hängt in Abschnitt A.5 an.

Das Ziel des Fragebogens war es, die Erfahrungen der Probanden mit Hinblick auf mehrere Aspekte zu erfassen: den Einfluss der Verwendung von KI-Werkzeugen auf das SE und auf die Zufriedenheit der Entwickler, sowie die Frage nach dem Umgang, in welchem KI-Werkzeuge das SE bereits heute un-

terstützen können. Abgefragt wurde der Erfahrungsstand der einzelnen Probanden (sowohl im SE generell, als auch im Umgang mit den Werkzeugen), die Anwendungsfälle und Einsatzgebiete für welche die Probanden konkrete Anwendung der Werkzeuge gefunden haben, der Maß des Erfolges, die Qualität und die Effizienz dieses Einsatzes. Außerdem wurde die Zufriedenheit und die Lessons Learned der Probanden dokumentiert. Die statistische Aussagekraft der Ergebnisse ist durch die geringe Zahl an Probanden beschränkt. Deshalb zielt der Fragebogen auf die qualitative Erhebung von Erfahrung ab. Zum Einsatz kommt zum einen eine leicht zu vergleichende 5-Punkte-Skala — 1 = starke Ablehnung, 2 = Ablehnung, 3 = neutral, 4 = Zustimmung und 5 = starke Zustimmung. Zum Anderen liegt der Fokus jedoch auf Freitextantworten, um die Erfahrungen der Probanden tiefergehend erfassen zu können. Darüber hinaus gab es einen interaktiven Teil, in welchem der Interviewer und der Proband gemeinsam die Antworten der Probanden in eine bestimmte Struktur überführten. Im Detail werden die Fragen im nachfolgenden Abschnitt 4.2 besprochen.

4.2 INHALT UND ANTWORTEN DES FRAGEBOGENS

Dieser Abschnitt stellt den Inhalt des Fragebogens und die Antworten der Probanden vor. Für jede Frage wird zunächst die Frage selbst und die Zielsetzung hinter ihr vorgestellt. Danach werden die Antworten der Probanden auf die Frage vorgestellt und eingeordnet. Eine weiterführende Interpretation der Ergebnisse erfolgt im Anschluss in Abschnitt 4.4. Der Übersicht halber gliedert sich dieser Abschnitt in drei Teile: Metadaten (Unterabschnitt 4.2.1), Konkrete Erfahrungen (Unterabschnitt 4.2.2) und Einordnung der Erfahrungen (Unterabschnitt 4.2.3). Diese Aufteilung ist lediglich ein formelles Mittel der vorliegenden Arbeit. Sie fand nach der Untersuchung statt und kam in den Interviews nicht zum Einsatz.

4.2.1 *Metainformationen*

> **Frage 1 — Zur Person:**
> Wie lange arbeitest du bereits als Software Engineer?

Die Frage nach der Berufserfahrung der Probanden zielt darauf ab, Kontext für die weiteren Antworten zu sammeln, um diese besser einordnen zu können. Dass die Berufserfahrung einen Einfluss auf den Umgang mit KI-Werkzeugen im SE hat, ist denkbar. Die Berufsbezeichnung „Software Engineer“ und welche Tätigkeiten genau dazu zählen wurde nicht näher definiert, sondern unterlag der individuellen Interpretation der Probanden. Im Schnitt haben die Probanden knapp über 7 Jahre Erfahrung als Software Engineers. Das Spektrum reicht von einem halben bis hin zu 40 Jahren. Die Standardabweichung liegt bei circa 8,5 Jahren und ist damit sogar höher als die durchschnittliche Erfahrung. Die hohe Diversität im Erfahrungsstand der Proban-

den ist ein Indikator für die Belastbarkeit und große Themenabdeckung der Ergebnisse.

Frage 2 — Erfahrungsstand:

a. Ich verwende das KI-Werkzeug seit *weniger als 1 Woche* / *weniger als 1 Monat* / *1 bis 3 Monate* / *mehr als 3 Monaten*.

b. Ich schätze mich selbst im Umgang mit dem KI-Werkzeug als *sehr ungeübt* / *ungeübt* / *etwas erfahren* / *erfahren* / *sehr erfahren* ein.

c. Mit welchen Programmiersprachen, IDEs und/oder Frameworks hast du das Werkzeug verwendet?

d. In welchem Projektkontext? (Kundenprojekt, privat, ...) Wie lange bist du bereits in diesem Projektkontext aktiv?

Frage 2 besteht aus vier Teilfragen a bis d. Sie zielt darauf ab, den Erfahrungsstand des jeweiligen Probanden im Umgang mit einem bestimmten KI-Werkzeug zu erfassen und damit ebenfalls Meta-Informationen für weitere Antworten zu sammeln. Frage 2 wurde für jedes erprobte Werkzeug ein Mal gestellt, sodass Probanden die mehrere Werkzeuge erprobten, die Frage mehrfach beantworteten. Frage 1 war die einzige Frage, welche kein konkretes Werkzeug im Fokus hatte. Alle weiteren Fragen des Fragebogens folgen dem Muster von Frage 2 und wurden im Falle mehrerer KI-Werkzeuge wiederholt gestellt.

Teilfrage a erfasst den Zeitraum, in welchem eine Probandin ein Werkzeug erprobt hat. Zur Auswahl stehen vier Antwortmöglichkeiten, welche jeweils mit Slashes (/) voneinander getrennt dargestellt sind. Die Antwortmöglichkeiten bilden vier größer werdende Zeitintervalle ab, da untersucht werden soll, inwiefern der Lerneffekt in Verwendung der Werkzeuge mit der Zeit abnimmt. Die Beschränkung auf vier diskrete Intervallen vereinfacht dabei außerdem die Frage. Letztendlich wurden die Intervalle willkürlich ausgewählt, da sie nur eine grobe Einordnung darstellen. Insgesamt wurde ChatGPT von 2 Probanden *weniger als 1 Woche*, 0 *weniger als 1 Monat* (aber mehr als 1 Woche), 5 im Intervall *1 bis 3 Monate* und 9 Probanden für *mehr als 3 Monate* verwendet. GitHub Copilot wurde von 2 Probanden *weniger als 1 Woche*, 1 *weniger als 1 Monat* (aber mehr als 1 Woche), 8 im Intervall *1 bis 3 Monate* und 0 Probanden für *mehr als 3 Monate* verwendet.

Teilfrage b hat ein ähnliches Format wie a und verlangt die Auswahl einer von fünf vorgefertigten Antwortmöglichkeiten. Die Selbsteinschätzung der Probanden sah wie folgt aus: Im Umgang mit ChatGPT haben sich 0 Probanden als *sehr ungeübt*, 3 als *ungeübt*, 4 als *etwas erfahren*, 8 als *erfahren* und 1 Proband als *sehr erfahren* eingeschätzt. Mit GitHub Copilot haben sich 0 Probanden als *sehr ungeübt*, 4 als *ungeübt*, 5 als *etwas erfahren*, 2 als *erfahren* und 0 Probanden als *sehr erfahren* eingeschätzt.

Teilfrage c verlangt als Freitextantwort eine Liste der Programmiersprachen, Entwicklungsumgebungen und Frameworks, im Kontext derer die Probanden ihre Erfahrung sammelten. Über beide Werkzeuge hinweg waren die Pro-

banden im Kontext von insgesamt 56 verschiedenen Programmiersprachen, Frameworks und Entwicklungsumgebungen aktiv. Am häufigsten wurden dabei die folgenden genannt: Typescript, Angular, Visual Studio Code, Java, C#, HTML, Python, JetBrains IntelliJ und Vue.js. Tabelle 4.1 zeigt die Quantität der Nennungen je KI-Werkzeug. Eine vollständige Liste aller genannten Programmiersprachen, Frameworks und Entwicklungsumgebungen, sowie deren Quantität hängt in Abschnitt A.2 an. Das sehr breite Antwortspektrum lässt sich als sehr repräsentativ für das Software Engineering im Allgemeinen interpretieren. Damit sind die Ergebnisse nicht nur auf die betrachteten, sondern auch auf weitere oder zukünftige Programmiersprachen und Frameworks anwendbar. Da insgesamt jedoch nur wenige Antworten pro individueller Programmiersprache, Framework etc. gegeben wurden, können keine belastbaren Aussagen über den Effekt der Werkzeuge in bestimmten Umfeldern getroffen werden.

Programmiersprache, Framework oder Entwicklungsumgebung	ChatGPT	GitHub Copilot	Nennungen insgesamt
Typescript	6	8	14
Angular	6	5	11
Visual Studio Code	3	8	11
Java	7	2	9
C#	4	4	8
HTML	4	2	6
Python	3	2	5
JetBrains IntelliJ	3	2	5
Vue.js	3	2	5

Tabelle 4.1: Die Anzahl Nennungen der häufigsten verwendeten Programmiersprachen, Frameworks und Entwicklungsumgebungen. Dargestellt je KI-Werkzeug und insgesamte Nennungen.

Teilfrage d erfasst den Projektkontext, in welchem die Probanden die Werkzeuge verwendet haben. In erster Linie wird zwischen professionellen (im Auftrage von Kunden oder Firmenintern) und privaten (nicht-professionellen) Projekten unterschieden. Hintergrund dieser Teilfrage ist die Erwartung, professionelle Projekte seien näher an Realbedingungen für das Software Engineering als private Projekte. Das macht die Erfahrungen im Kontext von professionellen Projekten besonders aussagekräftig. Darüber hinaus erfasst die Teilfrage, wie versiert ein Proband im jeweiligen Projektkontext ist. Beide Informationen wurden in Form von Freitext erfasst und im Nachgang händisch auf eine von vier Kategorien reduziert: Nicht in diesem Kontext verwendet, Einsteiger, Fortgeschritten und Experte. In diese Abbildung flossen die Erfahrungsdauer und eine Selbsteinschätzung der Probanden im jeweiligen Projektkontext ein. Grob orientierten wurde sich dabei an den willkürlich gewählten Intervallen von unter einem Jahr als Einsteiger, 1 bis 3 Jahre als Fort-

geschritten und mehr als 3 Jahre als Experte orientiert. Insgesamt haben 12 Probanden mit ChatGPT Erfahrung im Kontext von professionellen Projekten gesammelt, 13 im Kontext von privaten Projekten. Davon waren 3 Einsteiger, 3 Fortgeschrittene und 6 Experten im professionellen Kontext und 4 Einsteiger, 4 Fortgeschrittene und 5 Experten im privaten Kontext. GitHub Copilot haben 9 in professionellem und 3 in privatem Kontext erprobt. Darunter 1 Einsteiger, 3 Fortgeschrittene und 5 Experten im professionellen Kontext und 1 Einsteiger, 1 Fortgeschrittene und 1 Experten im privaten Kontext. Es sei darauf hingewiesen, dass die Kontexte sich nicht gegenseitig ausschließen. Eine Probandin könnte ein Werkzeug in beiden Kontexten eingesetzt haben.

> **Frage 3 — Einarbeitung:**
> Hast du dich in die KI-Werkzeuge gezielt eingearbeitet? Wie hat der Einstieg stattgefunden? Wie hast du gelernt mit dem Werkzeug umzugehen?

Frage 3 beleuchtet die Art der Einarbeitung zwischen Probanden und Werkzeugen. Hintergrund ist ebenfalls die Sammlung von Meta-Informationen, um weitere Antworten besser einordnen zu können. Frage 3 wurde für jedes erprobte Werkzeug ein Mal gestellt und wurde von Probanden, die mehrere Werkzeuge erprobten, mehrfach beantwortet. Auch hier handelt es sich bei den Antworten um Freitextantworten, die im Nachhinein händisch auf zwei Kategorien reduziert wurden: Gezielt eingearbeitet und Learning by doing. Insgesamt gaben von den 16 Probanden, die ChatGPT erprobten, nur 5 an sich gezielt in den Umgang mit dem Werkzeug eingearbeitet zu haben. Bei GitHub Copilot war es nur 1 von 11 Probanden, der angab sich gezielt einarbeitet zu haben. Es sticht deutlich hervor, dass die Mehrheit der Probanden sich nicht gezielt eingearbeitet hat, sondern sich organisch im Arbeitsfluss mit den Werkzeugen vertraut machte.

4.2.2 *Konkrete Erfahrungen*

> **Frage 4 — Anwendungsfälle/Einsatzgebiete:**
> Wo und wie hast du das KI-Werkzeug eingesetzt? Wo verlief es besonders erfolgreich/-los? Warum?

Frage 4 steigt in den Kern des Fragebogens, die konkreten Erfahrungen mit den KI-Werkzeugen, ein. Ziel ist es, genau zu verstehen, in welchen Anwendungsfällen die Probanden Erfahrungen gesammelt hatten und inwieweit die Unterstützung der Werkzeuge dabei hilfreich war. Frage 4 wurde ebenfalls ein Mal je Werkzeug gestellt. Das Antwortformat war Freitext. Während die Probanden ihre Erfahrungen schilderten, dokumentierte der Interviewer vor den Augen der Probanden textuell auf dem Fragebogen. Um die gesammelten Erfahrungen aller Probanden strukturiert erfassen zu können, werden die Antworten der Probanden auf Frage 4 in Frage 5 aufgegriffen. Die Antworten auf Frage 4 sind somit nur ein Zwischenergebnis, auf welches nicht näher eingegangen wird.

Frage 5 — Aufgabengebiete nach *BeST*:
Abbildung 3.1 zeigt das *BeST-Foundation Framework*. Wo lassen sich deine Erfahrungen einordnen?

Frage 5 dient dem Zweck, die in Frage 4 erfassten Erfahrungen zu strukturieren. Als zugrunde liegende Struktur dient dabei ein allgemeiner Softwareentwicklungsprozess, wie er in *BeST* dargestellt ist. Wie in Unterabschnitt 3.2.1 vorgestellt, gliedert sich *BeST* in sieben Domänen und darunterliegende Aufgabenbereiche. In Vorbereitung auf die Interviews wurde die *BeST*-Struktur um eine dritte Ebene unter den Aufgabenbereichen erweitert: die Anwendungsfälle oder auch Use-Cases. Diese dritte Ebene beinhaltet beispielsweise in der Domäne *4. Implementierung* unter dem Aufgabenbereich *4.3 Code schreiben* Anwendungsfälle wie *Refactoring & Code Optimierung* oder *Lernen einer neuen Programmiersprache/Framework schreiben*. Eine erste Zahl dieser Anwendungsfälle wurde auf Basis der drei vorab geführten Interviews gesammelt. Im Verlauf der Interviews wurde diese Liste in Zusammenarbeit mit den Probanden um weitere Anwendungsfälle erweitert. Dabei wurde stets dann ein neuer Anwendungsfall aufgenommen, wenn ein Aspekt der Erfahrung eines Probanden nicht durch bereits existierende Anwendungsfälle abgedeckt wurde, oder nicht präzise genug abgedeckt wurde. Es wurden nur Anwendungsfälle berücksichtigt, welche das Software Engineering betrafen. Zwischen den vorab gesammelten und den während den Interviews aufgenommenen Anwendungsfällen wird nicht unterschieden. Die im Nachfolgenden verwendeten Formulierungen, ein Anwendungsfall wurde „von Probanden erprobt“ bzw. ein Anwendungsfall wurde „im Rahmen der Untersuchungen identifiziert“ beinhalten keine Aussage über die Herkunft der Anwendungsfälle. Frage 5 wurde ebenfalls ein Mal je Werkzeug bearbeitet. Dabei wurde Frage 5 absichtlich im Anschluss an Frage 4 gestellt, um die ursprünglichen Antworten der Probanden auf Frage 4 nicht durch die Präsentation vieler Anwendungsfälle zu verfälschen. Tatsächlich äußerte eine handvoll Probanden während Frage 5 erstaunen über einige Anwendungsfälle, welche sie noch nicht ausprobiert hatten.

Zur Dokumentation der Antworten verließ der Interviewer gemeinsam mit den Probanden den Fragebogen und legte eine Tabelle auf, die einen Überblick über die Anwendungsfälle gab. Abbildung 4.1 zeigt einen Ausschnitt dieser Tabelle. Die Probanden sahen lediglich die Domänen, Aufgabengebiete und Anwendungsfälle in ihrer bevorzugten Sprache, nicht jedoch die Antworten der anderen Probanden. Auf Basis ihrer Antworten auf Frage 4, wurde mit den Probanden besprochen, ob sie die vorliegenden Anwendungsfälle erprobt hatten. Die Probanden vergaben Noten abhängig davon, wie erfolgreich oder erfolglos dies verlaufen war. Die Benotung geschah anhand einer Skala von 1 bis 5, wobei 1 für *deutlich erfolglos*, 2 für *erfolglos*, 3 für *durchwachsen*, 4 *erfolgreich* und 5 für *deutlich erfolgreich* stand. Teilweise wurden auch Noten zwischen diesen Werten zugelassen, etwa 3,5, um die Antworten der Probanden präziser abbilden zu können. Neben dieser Benotung hatten die Probanden Gelegenheit, Kommentare als Freitext an Noten anzuhängen. Im Zuge von Frage 5 erweiterten beide Parteien in Zusammenarbeit die Liste um weitere

	Domänen	Aufgabenbereich	Anwendungsfälle	Proband A ChatGPT	Proband B ChatGPT	Proband B Phind	Proband B Copilot	Proband C ChatGPT	Proband C Copilot	Proband D ChatGPT	Proband D Copilot	Proband E ChatGPT	Proband E Copilot
50													
51		4.1 Implementierung als Disziplin verstehen											
52		4.2 Feindesign erstellen	Schnittstellen-Schema erstellen (z.b. JSON)					4					
53			Code schreiben (generieren)	5			4	3	4	3	4	4	5 (ir
54			Refactoring & Code Optimierung	5	3	3					1		
55			Debugging & Troubleshooting										
56			Fehlermeldungen Analysieren (und beheben)	5	4	4		5 (analysieren ja, beheben nein)		2		3	
57		4.3 Code schreiben	Code Snippets verstehen (erklären lassen)		5	5	4	4					
58	4. Implementierung		Lernen einer neuen Programmiersprache/Framework		5	5		5		3	2	4	
59			Coding mit domain-specific languages (Regex, SQL)					5		5 (Regex); 2 (SQL)			
60			Ressourcen erstellen (z.b. SVG)										
61			Code konvertieren (Programmiersprachen, Version)		3	3							
62		4.4 Codequalität absichern	Code Review										
63		4.5 Code dokumentieren	Code kommentieren (inline)							4	3 bis 4 (nach Komplexität des zu kom		
64			Code kommentieren (out-of-code Dokumentation)										
65		4.6 Codequalität bewerten und Code übernehmen											
66													
67		5.1 Teststrategie definieren											
68		5.2 Test planen und vorbereiten											
69			Testdaten generieren/erstellen								5		
70	5. Test	5.3 Manuell testen	Testfälle identifizieren										
71		5.4 Automatisiert testen	Automatisierte Tests schreiben (Code)					5 (Einzelne Assertions)					
72		5.5 Tests dokumentieren und auswerten	Testcode kommentieren (inline)										
73		5.6 Testqualität sicherstellen											
74													
75		6.1 Entwicklung und Betrieb integrieren											
76		6.2 Infrastruktur entwickeln	Integration mit CI/CD, Docker, etc. aufsetzen							3 bis 4			
77	6. Infrastruktur & Betrieb	6.3 Prozesse automatisieren	Coding mit Infrastructre as Code										
78		6.4 Software betreiben											
79		6.5 Entwicklungsumgebung bereitstellen	Setup der Arbeitsumgebung (IDE, etc.)	5					3			4	
80													
81		7.1 Projektmanagement als Disziplin verstehen											
82		7.2 Projekt planen und kalkulieren	Antworten auf Ausschreibungen erstellen (generieren)										
83		7.3 Projekt überwachen											
84		7.4 Umfang kontrollieren											

Abbildung 4.1: Auszug der Tabelle von Anwendungsfällen. Während des Interviews sagen die Probanden lediglich die Domänen, Aufgabengebiete und Anwendungsfälle in ihrer bevorzugten Sprache, nicht jedoch die Antworten der anderen Probanden.

Anwendungsfälle, welche die Probanden erprobt hatten. Durch dieses stetige Wachstum der Liste bekamen die zuerst interviewten Probanden weniger Anwendungsfälle präsentiert als diejenigen, deren Interviews zu einem späteren Zeitpunkt stattfanden. Zum Ende der Interviews wurden insgesamt 33 Anwendungsfälle für KI-Werkzeuge im SE in sechs der sieben Domänen identifiziert:

In *Domäne 1. Vorgehensstrategie* wurden keine Anwendungsfälle identifiziert. In *Domäne 2. Requirements & Analyse* wurden 3 Anwendungsfälle identifiziert: Personas erstellen, Verstehen technischer Anforderungen und Verstehen von Implikationen und Abhängigkeiten zwischen Requirements. In *Domäne 3. Architektur & Design* waren es 9 Anwendungsfälle: Betriebsdokumentation verstehen, Architekturpatterns & -Stile verstehen, Ideen zur Anforderungsberücksichtigung entwickeln, Potenzialanalyse anhand priorisierter Qualitätsmerkmale, Komponentenschnitt erarbeiten, Trade-offs unterschiedlicher Lösungen vergleichen, Architektur textuell dokumentieren, Unified Modeling Language (UML)-Diagramme erstellen und ER-Modelle zu SQL Create-Statements erstellen. In *Domäne 4. Implementierung* wurden mit 13 die meisten Anwendungsfälle identifiziert: Schnittstellenschema erstellen, Code schreiben, Refactoring & Code Optimierung, Debugging & Troubleshooting, Fehlermeldungen analysieren, Codeabschnitte verstehen, Lernen neuer Programmiersprachen, Coding mit Domain-Specific Languages, Ressourcen erstellen, Code konvertieren, Code Review, Code kommentieren (inline) sowie Code kommentieren (out-of-code). In der *Domäne 5. Test* waren es 4 Anwendungsfälle: Testdaten erstellen, Testfälle identizifieren, Automatisierte Tests schreiben und Testcode kommentieren. In *Domäne 6. Infrastruktur & Betrieb* wurden 3 Anwendungsfälle identifiziert: Integration mit Continuous Integration / Continuous Delivery (CI/CD), Setup der Arbeitsumgebung sowie Coding mit IaC. In der *Domäne 7. Projektmanagement* letztlich wurde 1 Anwendungsfall erfasst: Ausschreibungen beantworten.

Diese umfangreiche Liste von Anwendungsfällen ist ein Indikator für die große Bandbreite an Einsatzmöglichkeiten von KI-Werkzeugen im SE. Die einzelnen Anwendungsfälle werden in Abschnitt 4.3 im Detail vorgestellt, um den Umfang des aktuellen Abschnittes nicht zu sprengen. Dort finden sich ebenfalls die Anzahl der erprobenden Probanden, die durchschnittliche Bewertung sowie eine Analyse der Freitextkommentare je Anwendungsfall.

4.2.3 *Einordnung der Erfahrungen*

Dieser Unterabschnitt geht auf die einordnenden Fragen 6 bis 12 ein. Teilweise bedienen auch diese Fragen sich einer 5-Punkte-Skala, jedoch schließen alle einen Freitextanteil ein. Das Thema „KI-gestützte Softwareentwicklung mithilfe LLM-basierter Werkzeuge“ ist ein neues und bislang wenig erforschtes Forschungsfeld. Deshalb ist davon auszugehen, dass die Fragen nicht präzise genug formuliert sind, um die Erfahrungen der Probanden anhand von 5-Punkte-Skalen vollständig zu erfassen. Das macht die Freitextantworten besonders relevant für die Einordnung der Erfahrungen.

Um einen Überblick über den Freitext zu 16 Erfahrungen mit ChatGPT und zu 11 Erfahrungen mit GitHub Copilot zu gewinnen, wurden die Freitextantworten der Probanden zusammengefasst. Dabei wurden innerhalb der Antworten zu einer Frage und einem Werkzeug die Antworten der Probanden zusammengefasst, welche sich inhaltlich ähnelten und notiert, wie viele Probanden eine jeweilige Antwort gegeben hatten. Die auf diese Weise zusammengefassten Antworten, welche von mehr als einem Probanden genannt wurden, werden im Folgenden vorgestellt. Einzelne Antworten werden aufgrund des großen Umfangs der Antworten und deren beschränkter Aussagekraft nicht diskutiert.

Zur Auswertung der Antworten auf der 5-Punkte-Skala werden die Antwortmöglichkeiten als Zahlen von 1 bis 5 interpretiert. Dabei ist die 1 die negativste, 3 die neutrale und die 5 die positivste Antwortmöglichkeit. So können Durchschnitt und Standardabweichung der Bewertung berechnet werden. Es wurde sich absichtlich gegen die Verwendung von Konfidenzintervallen entschieden, da die Datenmenge zu klein ist, um diese sinnvoll zu verwenden. Als Konsequenz dieser Methodik sind statistisch verlässliche Aussagen nicht möglich, wie etwa ein Intervall, in welchem sich mit hoher Wahrscheinlichkeit die tatsächlichen Einschätzungen der Probanden befindet. Die im Mittel vergebene Note ist lediglich ein Indikator für die Einschätzung der Probanden, während die Standardabweichung ein Indiz für die Streuung der Daten ist. So lässt eine durchschnittliche Note von $3,5$ bei einer Standardabweichung von $0,5$ zwar eine tendenziell positive Erfahrung der Probanden vermuten, jedoch ist dies kein Garant dafür, dass alle Antworten zwischen 3 und 4 liegen. Auch dass alle Probanden diesen Aspekt als positiv wahrgenommen haben, lässt sich nicht ableiten.

> **Frage 6 — Qualität:**
> Durch die Verwendung des KI-Werkzeuges ist die Qualität meiner Arbeit *deutlich schlechter* / *schlechter* / *unverändert* / *höher* / *deutlich höher*. (Qualität kann vieles bedeuten: Codequalität, umfassendere Anforderungsberücksichtigung, ...)
>
> Welche Tätigkeiten sind betroffen? Warum?

Frage 6 erfasst, ob und inwiefern die Probanden im Umgang mit den Werkzeugen Unterschiede in der Qualität ihrer Arbeit feststellten. Sie besteht aus der Auswahl einer von fünf vorgefertigten Antwortmöglichkeiten und aus einem Freitextanteil. Auf die Ambivalenz des Begriffs „Qualität“ im Kontext von Software Engineering wurde explizit hingewiesen. Letztendlich unterlag die Interpretation des Begriffs jedoch jedem Probanden selbst. 3 Probanden sahen die Qualität ihrer Arbeit unter Verwendung von ChatGPT als *unverändert*, 11 Probanden als *höher* und 1 Proband als *deutlich höher*. Ein Proband machte aufgrund seiner geringen Erfahrung mit ChatGPT keine Aussage. Mit einer durchschnittlichen Bewertung von $\approx 3,87$ und einer Standardabweichung von $\approx 0,52$ scheinen die Probanden tendenziell eine Steigerung der Qualität ihrer Arbeit festzustellen. Unter Verwendung von GitHub Copilot 1 Proband

die Qualität seiner Arbeit als *schlechter*, 6 Probanden als *unverändert* und 4 Proband als *höher*. Im Kontrast zu ChatGPT schneidet die Erfahrung mit Copilot hier mit einer durchschnittlichen Bewertung von $\approx 3,27$ schlechter ab. Mit einer Standardabweichung von $\approx 0,65$ liegen die Antworten außerdem ein wenig weiter auseinander. Insgesamt scheinen beide Werkzeuge die Qualität der Arbeit der Probanden im Mittel zu verbessern, wobei ChatGPT besser abschneidet.

In den Freitextantworten zu ChatGPT wurden die folgenden Punkte von mehr als einem Probanden genannt: Zwei Probanden sahen die Qualität besonders dann erhört, wenn sie mit Umgebungen und Programmiersprachen arbeiteten, mit denen sie nicht vertraut waren. Vier Probanden gaben an, der Chatbot hätte ihnen Ideen oder Ansätze vorgeschlagen, an die sie sonst nicht gedacht hätten. Drei Probanden fiel die Unterstützung beim Lernen positiv auf. Darüber hinaus hätte der Chatbot Fragen beantworten können, welche sie sonst Kollegen hätten stellen müssen. Vier Probanden nahmen ChatGPT als eine Alternative zu gängigen Websuchmaschinen war, welche teilweise schneller Ergebnisse, oder sogar bessere Ergebnisse lieferte als die Suchmaschine. Zwei Probanden wiesen darauf hin, dass die Qualität nur dann gleich blieb, wenn die Ausgaben des Chatbots genau geprüft und nicht blind übernommen wurden.

In den Freitextantworten zu GitHub Copilot wurden die folgenden Punkte von mehr als einem Probanden genannt: Drei Probanden gaben an, die Qualität der mithilfe von Copilot geschriebenen Dokumentation und Kommentare (sowohl inline als auch out-of-code) sei höher. Fünf Probanden sahen die Codequalität als unverändert, während zwei weitere Probanden darauf hinwiesen, dass die Qualität nur dann auf demselben Niveau blieb, wenn die Vorschläge von Copilot vor der Übernahme gewissenhaft geprüft wurden. Zwei Probanden erlebten die Unterstützung von Copilot als schlechter, je mehr domänenspezifisches Wissen der bearbeitete Abschnitt beinhaltete.

> **Frage 7 — Effizienz:**
> Mithilfe der Unterstützung durch das KI-Werkzeug arbeitete ich *deutlich langsamer* / *langsamer* / *mit unveränderter Geschwindigkeit* / *schneller* / *deutlich schneller*.
>
> Welche Tätigkeiten sind betroffen? Warum?

Frage 7 liegt analog zu Frage 6, zielt jedoch auf die Effizienz bzw. Geschwindigkeit der Arbeit ab. Sie besteht ebenfalls aus einer Auswahl einer vorgefertigten Antwortmöglichkeit und einem Freitextanteil. Unter Verwendung von ChatGPT sahen 5 Probanden die Geschwindigkeit ihrer Arbeit als *unverändert*, 8 Probanden als *schneller* und 3 Probanden als *deutlich schneller*. Die durchschnittliche Bewertung von $\approx 3,88$ und die Standardabweichung von $\approx 0,72$ lassen auf einen geschwindigkeitssteigernden Effekt schließen. Für GitHub Copilot bewerteten 4 Probanden ihre Arbeit als *unverändert* und 7 Probanden als *schneller*. Die durchschnittliche Bewertung von $\approx 3,64$ und die Standardabweichung von $\approx 0,50$ lassen auf einen ähnlich positiven Effekt schließen.

In den Freitextantworten zu ChatGPT wurden die folgenden Punkte von mehr als einem Probanden genannt: Sieben Probanden sahen ihre Geschwindigkeit beim Schreiben von Code durch die Unterstützung von ChatGPT gesteigert. Insbesondere wurde dabei Boilerplate Code erwähnt und dass der Chatbot vor allem bis hin zum Erreichen einer ersten lauffähigen Version effektiv unterstützt. Drei Probanden gaben an, die Analyse von Fehlermeldungen mithilfe von ChatGPT sei besonders von dem positiven Effekt betroffen. Sechs Probanden sahen den Geschwindigkeitszuwachs dadurch zurückgehalten, dass in der Zusammenarbeit mit dem Chatbot häufig mehrere Iterationen von Prompts, viel Kommunikation und genaue Prüfung der Ausgaben notwendig waren. Teilweise drehe sich die Unterhaltung mit dem Chatbot im Kreis, ohne dabei zu einem Ergebnis zu kommen. Vier Probanden nannten ChatGPT eine Alternative zu gängigen Websuchmaschinen mit vergleichbarer Geschwindigkeit. Dabei liefere der Chatbot Wissen, welche sie im Fall der Websuche selbst händisch filtern und aufbereiten müssten.

In den Freitextantworten zu GitHub Copilot wurden die folgenden Punkte von mehr als einem Probanden genannt: Fünf Probanden sahen einen besonderen Geschwindigkeitszuwachs bei dem Schreiben von Dokumentation und Kommentaren (sowohl in-line als auch out-of-code). Sieben Probanden gaben an, das Schreiben von repetitivem Code sei besonders betroffen. Laut sieben Probanden wurde dieser positive Effekt jedoch durch die Notwendigkeit zurückgehalten, die Vorschläge zu überprüfen, überarbeiten oder zu verwerfen.

> **Frage 8 — Fähigkeiten/Lerneffekt:**
> Das KI-Werkzeug erlaubt es mir, Dinge zu tun, die ich ohne nicht könnte. Ich *widerspreche deutlich* / *widerspreche* / *sehe das neutral* / *stimme zu* / *stimme deutlich zu*.
>
> Welche Fähigkeiten sind betroffen? Warum?

Frage 8 konfrontiert die Probanden mit einer Aussage, welche sie ähnlich zu den Fragen davor durch die Auswahl einer von fünf vorgefertigten Antwortmöglichkeit und einem Freitextanteil bewerten sollten. Hintergrund ist die Annahme, KI-Werkzeuge hätten das Potenzial, die Fähigkeiten von Softwareentwicklern zu erweitern. Basierend auf der Verwendung von ChatGPT gaben 2 Probanden *deutlichen Widerspruch* an, 3 Probanden *widersprachen*, 2 Probanden *sahen das neutral*, 7 Probanden *stimmten zu* und 2 Probanden *stimmten deutlich zu*. Mit einer durchschnittlichen Bewertung von $3,25$ und einer Standardabweichung von $\approx 1,29$ sind die Probanden sich hier besonders uneinig. Bei GitHub Copilot gaben 4 Probanden *deutlichen Widerspruch* an, 3 Probanden *widersprachen*, 2 Probanden *sahen das neutral*, 1 Proband *stimmten zu* und 1 Proband *stimmte deutlich zu*. Die durchschnittliche Bewertung liegt bei $\approx 2,27$, die Standardabweichung bei $\approx 1,35$. Auch bei Copilot herrscht große Uneinigkeit unter den Probanden, wobei die Bewertung deutlich niedriger ausfällt als bei ChatGPT. Das lässt auf eine ablehnende Bewertung der Aussage im Fall von Copilot schließen. Frage 8 hat für beide Werkzeuge die höchste Standardabweichung des gesamten Fragebogens und ist die einzige Frage, bei welche

die Probanden als Gruppe das gesamte Antwortspektrum ausgeschöpft haben.

In den Freitextantworten zu ChatGPT wurden die folgenden Punkte von mehr als einem Probanden genannt: Sieben Probanden sahen die Möglichkeit, mithilfe des Chatbots in neuer Technologie oder Programmiersprache arbeitsfähig zu werden, ohne sofort alles verstehen zu müssen. Vier Probanden gaben an, die Unterstützung von ChatGPT hätte ihnen beim Lernen einer neuen Programmiersprache geholfen. Ebenfalls von vier Probanden explizit erwähnt wurde der Eindruck, der Chatbot helfe nur dabei Informationen zu finden, ohne die eigenen Fähigkeiten dabei zu verbessern.

In den Freitextantworten zu GitHub Copilot wurden die folgenden Punkte von mehr als einem Probanden genannt: Zwei Probanden fiel positiv auf, dass Copilot zu Lösungen inspirieren würden, derer man sich zuvor nicht bewusst war. Zwei weitere Probanden sehen die Inspiration zu neuen Lösungen zwar vorhanden, jedoch nur in unvertrauten Domänen als Mehrwert. Fünf Probanden gaben an, Copilot könne die Implementierung nur unterstützen, nachdem ein erster Schritt bereits getan sei (etwa erste Zeilen Code bereits geschrieben sein) und würde so nicht zu einem Fähigkeitsgewinn beitragen. Zwei Probanden beschrieben die Vorschläge des Werkzeugs lediglich als eine Alternative zur herkömmlichen Websuche und somit nicht als Erweiterung der eigenen Kompetenzen. Dabei würde ein Kommentar im Code verwendet werden, um zu recherchierende Funktionalität zu beschreiben. Nachdem der Cursor in die nächste Zeile bewegt wird, könnte Copilot diese Funktionalität vorschlagen. Ebenfalls von zwei Probanden wurde angegeben, die Vorschläge von Copilot wären teilweise nicht hilfreich gewesen und hätten Anti-Patterns beinhaltet.

> **Frage 9 — Potenziale:**
> Bei welchen Aufgaben siehst du Potenzial für das KI-Werkzeug? Basierend auf deiner Erfahrung, welche Aufgabengebiete scheinen in Zukunft in Reichweite des Werkzeugs?

Frage 9 dreht sich nicht wie die anderen Fragen auf den Einsatz von KI-Werkzeugen in der Gegenwart. Ihr Ziel ist es, auf Basis der Erfahrung der Probanden eine weitere Einschätzung über das Potenzial von dem Einsatz LLM-basierte Werkzeuge im SE zu gewinnen, welche die Potenzialanalyse aus Kapitel 3 um eine Prognose in die Zukunft erweitert. Das Antwortformat ist Freitext, der wie üblich vom Interviewer vor den Augen der Probanden auf Basis deren verbalen Äußerungen dokumentiert wurde. Eine Analyse der Antworten erfolgt in Unterabschnitt 4.4.3.

Für die Arbeit mit ChatGPT wurden folgende Punkte von mehr als zwei Probanden als Potenzial gesehen: Drei Probanden sahen eine Integration in die IDE als ein Potenzial den Workflow mit dem Werkzeug zu verbessern. Zwei Probanden sahen das Potenzial, dass neue Mitarbeiter in Zukunft mithilfe des Chatbots eingearbeitet werden könnten. Sechs Probanden hielten es für wahrscheinlich, dass ein weiterentwickeltes, kompetenteres Sprachmodell den Mehrwert, die Effizienz der Unterstützung ChatGPT's weiter steigern

könne. Zwei Probanden gaben das Testen von Code als ein Potenzial an, insofern der Chatbot einen größeren Kontext verarbeiten könne. Vier Probanden sahen umfassendere Unterstützung in der Softwarearchitektur als ein Potenzial in der Zukunft. Drei Probanden äußerten die Vermutung, LLM-basierte Chatbots könnten einen Paradigmenwechsel im SE einleiten, indem die Engineers hauptsächlich Code reviewen und sich auf Requirements-Engineering/-Management fokussieren. Drei Probanden hielten es für wahrscheinlich, dass die Unterstützung des Chatbots auch in Zukunft auf eine handvoll Anwendungsfälle beschränkt bleibt. Weitere drei Probanden sahen zukünftige Verbesserungen von ChatGPT durch einen Mangel an Kreativität und echtem fachlichen Verständnis zurückgehalten.

Für die Unterstützung von GitHub Copilot wurden die folgenden Punkte von zwei oder mehr Probanden genannt: Sechs Probanden sahen Potenzial für generell sinnvollere Vorschläge, welche die Unterstützung des Werkzeugs noch effizienter machen. Zwei Probanden hielten das Verbesserungspotenzial von Copilot durch fehlendes Domänenwissen und fachlichem Verständnis limitiert.

> **Frage 10 — Zufriedenheit & Well-being:**
> Ich werde das KI-Werkzeug in meinem SE-Alltag *definitiv nicht* / *nicht* / *vielleicht* / *weiterhin* / *definitiv weiterhin* einsetzen.
>
> Wie hast du die Arbeit mit diesen Werkzeugen empfunden? Komfortabel? Anstrengend? Vergleich von Werkzeugen untereinander?

Frage 10 bestand aus der Auswahl einer von fünf vorgefertigten Antwortmöglichkeit und einem Freitextanteil. Ziel der Frage ist, ein Verständnis für die Zufriedenheit der Probanden im Umgang mit den Werkzeugen zu gewinnen. Neben technischen Aspekten stellt das Wohlbefinden der Probanden ebenfalls einen wichtigen Faktor für die Evaluation der Werkzeuge dar. Die Verwendung von ChatGPT planen 2 Probanden *vielleicht*, 2 Probanden *weiterhin* und 12 Probanden *definitiv weiterhin* einzusetzen. Im Durchschnitt bewerteten die Probanden die Arbeit mit ChatGPT mit $\approx 4,63$ und einer Standardabweichung von $\approx 0,72$. Für GitHub Copilot planen 2 Probanden *vielleicht*, 3 Probanden *weiterhin* und 6 Probanden *definitiv weiterhin* einzusetzen. Bei Copilot war die durchschnittliche Bewertung $\approx 4,36$ mit einer Standardabweichung von $\approx 0,81$.

Zur Zufriedenheit im Umgang mit ChatGPT wurden folgende Punkte im Freitext mehr als einmal genannt: Neun Probanden nannten die Arbeit mit ChatGPT komfortabel, während zwei Probanden sie darüber hinaus sogar als unterhaltsam beschreiben. Vier Probanden sahen eine fehlende Verbindung von Code-Basis zum Chatbot als der Zufriedenheit abträglich. So sei die Arbeit mühsam, da in den Prompts der Ist-Zustand detailliert beschrieben werden müsse. Zwei Probanden fiel die Zugänglichkeit des Chatbots positiv auf. Man müsse kein Experte sein, um das Werkzeug effektiv nutzen zu können.

Zur Zufriedenheit im Umgang mit GitHub Copilot wurden folgende Aspekte von zwei oder mehr Probanden genannt: Sechs Probanden beschrieben die

Arbeit mit dem Werkzeug als komfortabel. Drei Probanden fiel die Integration von Copilot in den eigenen Workflow positiv auf. Das Werkzeug nehme Arbeit ab, ohne dabei neue Arbeit zu generieren. Drei Probanden sahen den Daten- und Geheimnisschutz als Hindernis für die Benutzung des Werkzeugs in weiteren Projekten. Außerdem müsse sich der Preis für Accso in einem vertretbaren Rahmen halten. Zwei Probanden gaben an, Usability-Probleme seinen ihrer Zufriedenheit abträglich. Es sei zu Konflikten zwischen der Autovervollständigung der IDE und Copilot gekommen. Teilweise seien die Antworten des Werkzeugs zeitversetzt gewesen und hätten im Falle von Internetproblem nur stellenweise funktioniert.

> **Frage 11 — Negative Aspekte:**
> Wo hat, in deinen Augen, das Werkzeug versagt? Was hat dir nicht gefallen?

Frage 11 soll den Probanden eine Möglichkeit geben, negativer Erfahrung Ausdruck zu verleihen, die sich noch nicht in den vorherigen Fragen niedergeschlagen hat. Auch der Blick auf die negativen Aspekte ist relevant, um ein vollständiges der Unterstützung der Werkzeuge zu erhalten. Das Antwortformat ist Freitext.

Bezüglich der Unterstützung durch ChatGPT wurden die folgenden Punkte von mehr als einem Probanden als negativ wahrgenommen: Vier Probanden fiel eine Unzuverlässigkeit des Chatbots negativ auf, der teilweise keine hilfreichen Ergebnisse liefere. Weitere vier nannten Halluzinationen von ChatGPT als negativen Aspekt. Drei Probanden sahen die zeitlich begrenzten Trainingsdaten als negativ an, da der Chatbot nicht über aktuelles Wissen verfüge.

Im Fall von Copilot sind zwei oder mehr Probanden die folgenden Aspekte negativ aufgefallen: Zwei Probanden gaben auch hier Halluzinationen als negativen Aspekt an. Es seien nicht existierende Funktionen vorgeschlagen worden. Weiteren zwei Probanden fiel negativ auf, dass Copilot an komplexen Codeabschnitten scheitere. Drei Probanden warnten, dass fehlerhafte und suboptimale Antworten des Werkzeugs sehr korrekt aussähen. Hier müsse der Output sehr genau geprüft werden, um Fehler zu vermeiden. Gerade unerfahrene Entwickler seien hier gefährdet. Zwei Probanden nannten Konflikte zwischen der Autovervollständigung der IDE und Copilot als negativen Aspekt.

> **Frage 12 — Lessons Learned:**
> Welche Tipps & Hinweise würdest du jemandem geben, der/die das Werkzeug zum ersten Mal nutzt?

Frage 12 zielt darauf ab, die Lessons Learned der Probanden zusammenzutragen. Die Antworten der Probanden dient als eine der Grundlagen für das in Kapitel 5 vorgestellte Nutzungskonzept und die Best Practices. Das Antwortformat ist ebenfalls Freitext.

Zu ChatGPT wurden folgende Lesson Learned von mehr als einem Probanden genannt: Zwei Probanden empfahlen, sich mit den Anwendungsfällen

des Werkzeugs vertraut zu machen. Um das volle Potenzial der Unterstützung nutzen zu können müsse man wissen, wo das Werkzeug unterstützen kann. Acht Probanden sahen einen Überblick über das Thema Prompt Engineering als relevant. Die Art und Weise wie Prompts gestellt würden, hätte einen großen Einfluss auf die Qualität der Antworten. Fünf Probanden erwähnten zudem, dass es wichtig sei, alle relevanten Informationen in den Prompt einzubauen. Dabei ginge es um den Kontext, etwa verwendete Programmiersprachen oder Libraries, sowie um fachliche Constraints und implizite Annahmen. Zwei Probanden empfahlen, den Chatbot im Prompt zu bitten, sich kurzzufassen. So seinen die Antworten übersichtlicher und man nutze das Token-Limit effizienter. Weitere zwei Probanden wiesen darauf hin, dass vor Einsatz des Werkzeugs genau geprüft werden müsse, ob der rechtliche Rahmen für dessen Einsatz mit Hinblick auf Daten- und Geheimnisschutz gegeben sei. Sechs Probanden nannten die Empfehlung, die Ausgaben von ChatGPT in jedem Fall kritisch zu prüfen. Fehler seien teilweise schwer zu erkennen. Drei Probanden empfahlen, in kleinen Abschnitten mit dem Chatbot zu arbeiten und nicht zu viel Code auf einmal zu generieren. So könnten die Ergebnisse besser bewertet und verstanden werden. Zwei Probanden nannten ein Bewusstsein über das Token-Limit als relevant. Sich dieser Einschränkung bewusst zu sein helfe, mit dem Chatbot zu arbeiten. Drei Probanden ermutigten dazu, eigenständig mit dem Werkzeug zu experimentieren und Erfahrungen zu sammeln.

Im Umgang mit GitHub Copilot wurden die folgenden Lesson Learned von zwei oder mehr Probanden genannt: Drei Probanden empfahlen, die Vorschläge von Copilot immer kritisch zu hinterfragen. Sie könnten fehlerhaft, veraltet oder suboptimal sein und trotzdem korrekt aussehen. Weitere drei Probanden ermutigten dazu, das Werkzeug selbst auszuprobieren und Erfahrungen im Umgang damit zu sammeln. Zwei Probanden empfahlen eine Einarbeitung in die von Copilot bereitgestellten Features, um sich aller Funktionen des Werkzeugs bewusst zu werden. Vier Probanden erwähnten, im Umgang mit Copilot seien nicht viele Lessons Learned zustande gekommen, da das Werkzeug sehr einfach zu benutzen sei.

4.3 ANWENDUNGSFÄLLE

Im Rahmen der Pilotuntersuchungen wurde eine Vielzahl von Anwendungsfällen durch die Probanden erprobt. Diese werden im Folgenden nach *BeST*-Domäne gegliedert vorgestellt, wobei die Relevantesten ausführlich diskutiert werden. Relevanz ist dabei wie folgt definiert: Je mehr Probanden einen Anwendungsfall erprobt haben und je besser die Benotung dabei ist, desto relevanter ist dieser. In dieser Arbeit werden die Anwendungsfälle genauer diskutiert, welche von mehr als einem Probanden je Werkzeug erprobt wurden und dabei eine Benotung von besser als Note 3 auf der 5-Punkte-Skala erhielten. Damit liegt der Fokus auf den Anwendungsfällen, bei denen — Einzelmeinungen ausgenommen — die Unterstützung der Werkzeuge besser als „unverändert“ wahrgenommen wurde. Im Rahmen der Diskussion relevanter

Anwendungsfälle ist jeweils aufgeführt, ob ein Anwendungsfall in Form einer Best Practice in das Nutzungskonzept aufgenommen wird. Eine Best Practice besteht hierbei aus der Kombination eines Anwendungsfalles mit einem Werkzeug, welches gewinnbringend in diesem eingesetzt werden kann. Vereinzelt werden Best Practices, in denen das Werkzeug einen besonders großen Mehrwert bietet, als **klare Empfehlung** bezeichnet. Diese Bezeichnung soll im Nutzungskonzept den Mehrwert hervorheben, ohne die Informationen aus diesem Abschnitt wiederholen zu müssen.

Ähnlich wie in Unterabschnitt 4.2.3 werden dabei als statistische Kennzahlen primär die durchschnittlich vergebene Note (Intervall 1 bis 5) und die Standardabweichung verwendet. Die Verwendung von Konfidenzintervallen ist dabei erneut aufgrund der kleinen Datenmenge nicht sinnvoll. Im Gegensatz zu Konfidenzintervallen sind die im Mittel vergebene Note und die Standardabweichung lediglich Indikatoren für die Bewertung der Probanden sowie die Streuung dieser. Statistisch verlässliche Aussagen lassen sich auf Basis dieser Kennzahlen nicht ableiten. Zusätzlich sind die im Mittel vergebene Note, die Standardabweichung sowie die Anzahl der Erfahrungen je Anwendungsfall und Werkzeug in den Abbildungen Abbildung 4.2, Abbildung 4.3, Abbildung 4.5 und Abbildung 4.4 grafisch dargestellt. Dabei stellt die obere Teilgrafik jeweils die im Mittel vergebene Bewertung dar. Falls dieser Durchschnitt durch mehrere, voneinander unterschiedlichen Bewertungen zustande gekommen ist, stellt eine schwarze Linie die Standardabweichung dar. Sie entspringt dem oberen Ende der Balken — der durchschnittlichen Bewertung — und reicht von dort um den Betrag der Standardabweichung in positive, als auch negative Y-Richtung. Ihre gesamte Länge entspricht also der doppelten Standardabweichung. Gibt es für einen Anwendungsfall mehrere Erfahrungen, welche mit derselben Note bewertet wurden, so ist keine schwarze Linie für diesen Anwendungsfall abgebildet. Die untere Teilgrafik stellt jeweils die Anzahl der Erfahrungen dar.

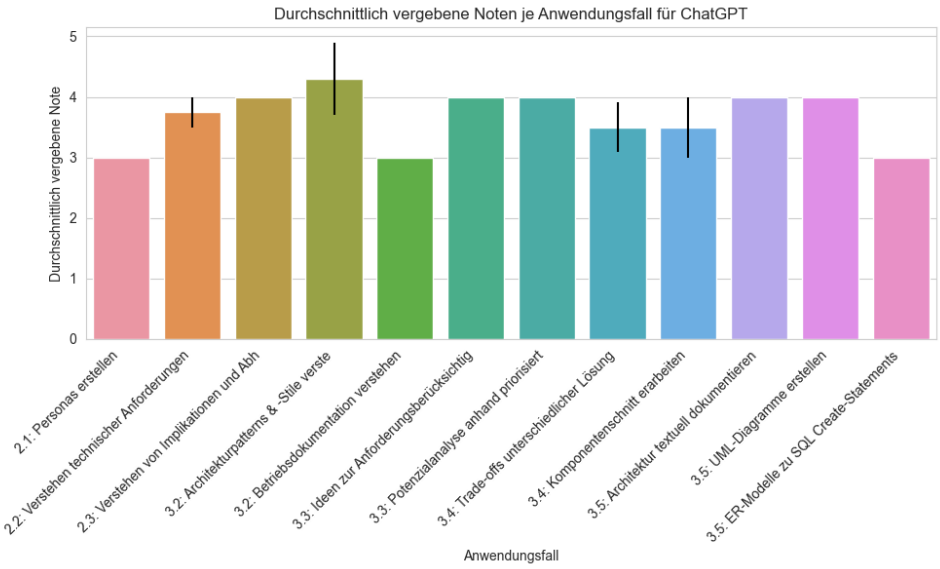
4.3.1 *Domäne 1. Vorgehensstrategie*

In der ersten Domäne *Vorgehensstrategie* haben wurden keine Anwendungsfälle identifiziert — weder auf Basis der drei vorab Interviews, noch mithilfe der Probanden. Es entstand der Eindruck, diese Domäne sei zu abstrakt und highlevel, als dass KI-Werkzeuge dort heute bereits zielführend eingesetzt werden können.

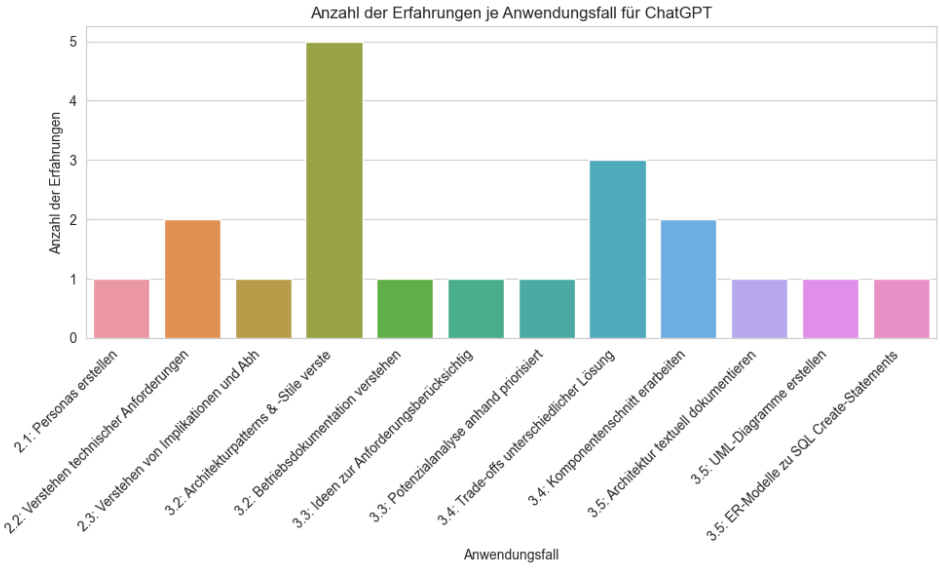
4.3.2 *Domäne 2. Requirements & Analyse*

In der zweiten Domäne *Requirements & Analyse* wurden drei Anwendungsfälle identifiziert.

Unter dem Aufgabenbereich *2.1 Nutzer verstehen* wurde *Personas erstellen* von einem Probanden unter Verwendung von ChatGPT erprobt und auf der 5-Punkte-Skala mit der Note 3 bewertet. Personas sind fiktive Charakterprofile, die typisch Nutzer eines Systems darstellen. Sie kommen unter anderem in



(a) Durschnittliche Noten und Standardabweichung



(b) Anzahl der Erfahrungen

Abbildung 4.2: Im Mittel vergebene Noten der Probanden für die Anwendungsfälle in den Domänen 2 und 3 unter Verwendung von ChatGPT.

der Softwareentwicklung zum Einsatz, um die Nutzer und deren Bedürfnisse besser verstehen und berücksichtigen zu können [Nie13, S. 2]. Das Werkzeug half hier bei der Erarbeitung von Personas durch den Vorschlag von Textbausteinen.

Unter 2.2 *Anforderungen verstehen* erprobten zwei Probanden den Anwendungsfall *Verstehen technischer Anforderungen* mithilfe von ChatGPT. Die durchschnittliche Bewertung war 3,75 bei einer Standardabweichung von 0,25. Der Chatbot scheint hier gewinnbringend eingesetzt worden zu sein. Hier wurde der Chatbot als Wissensressource verwendet, und nach möglichen Implikationen von Anforderungen gefragt. Nach der oben beschriebenen Metrik ist dieser Anwendungsfall relevant und wird deshalb genauer diskutiert. Aufgrund seines Trainings verfügt der Chatbot über ein umfassendes Weltwissen, was ihn zu einer nützlichen Wissensressource macht. Zwar haben die Probanden den Chatbot nicht im Kontext fachlicher Anforderungen erprobt, doch ist es denkbar, dass seine Unterstützung auch dort einen Mehrwert bietet kann. Der Chatbot könnte als ergänzende Wissensressource das Verständnis von Anforderungen erleichtern, indem er beim Brainstorming unterstützt sowie dabei, keine Aspekte zu übersehen. Dieser Anwendungsfall wird um die Unterstützung im Kontext fachlicher Anforderungen erweitert und als Best Practice mit dem Namen „Verstehen von Anforderungen“ in das Nutzungskonzept aufgenommen.

Unter 2.3 *Anforderungen verwalten* wurde *Verstehen von Implikationen und Abhängigkeiten zwischen Requirements* von einem Probanden unter Verwendung von ChatGPT erprobt und mit Note 4 benotet. Ähnlich wie bei dem vorhergegangenen Anwendungsfall wurde der Chatbot als Wissensressource verwendet. Trotz der guten Bewertung wird dieser Anwendungsfall nicht als relevant eingestuft, da er von nur einem Probanden erprobt wurde.

4.3.3 *Domäne 3. Architektur & Design*

In der dritten Domäne *Architektur & Design* haben die Probanden insgesamt neun Anwendungsfälle erprobt.

Unter dem Aufgabenbereich 3.2 *Orientierung geben* wurde der Anwendungsfall *Betriebsdokumentation verstehen* von einem Probanden mit ChatGPT erprobt und mit Note 3 bewertet. Dabei ging es um die Einarbeitung in eine vorhandene Betriebsdokumentation zum Einstieg in ein neues Projekt. Eine Herausforderung dabei war der Geheimnisschutz. Deshalb wurden dem Chatbot nur allgemeine Fragen gestellt, da der Projektkontext nicht an ihn übergeben werden durfte.

Ebenfalls unter 3.2 *Orientierung geben* erprobten 5 Probanden ebenfalls mithilfe von ChatGPT den Anwendungsfall *Architekturpatterns & -Stile verstehen*. Im Mittel bewerteten sie ihre Erfahrung mit der Note 4,30 bei einer relativ niedrigen Standardabweichung von 0,60. Dabei wurde der Chatbot explizit nicht als Diskussionspartner, sondern in erster Linie als Wissensressource verwendet. Die Probanden sahen diesen Anwendungsfall in Teilen durch die zeitlich limitierten Trainingsdaten begrenzt. Mit Bewertungen mehrerer Proban-

den, die im Schnitt besser als „erfolgreich“ ist, qualifiziert sich dieser Anwendungsfall als relevant und wird deshalb genauer diskutiert. Auch hier profitieren die Nutzer von dem umfangreichen Weltwissen des Chatbots. Zwar müssen die Ausgaben des Werkzeugs stets kritisch hinterfragt werden, doch scheint der Chatbot eine effektive Alternative zu Fachliteratur oder Onlinequellen zu sein. Ein großer Vorteil des Chatbots gegenüber herkömmlichen Quellen ist, dass er gezielt Fragen zu spezifischen Aspekten beantworten kann, ohne dass Nutzer große Informationsmengen händisch filtern und aufbereiten müssen. Trotzdem sind die Ausgaben von ChatGPT natürlich weniger verlässlich als Inhalte in der Fachliteratur oder vertrauenswürdigen Onlinequellen. Hier besteht die Abwägung zwischen Zeitersparnis und Verlässlichkeit. Dieser Anwendungsfall wird unter Verwendung von ChatGPT aufgrund der herausragenden Bewertung durch die Probanden als Best Practice und klare Empfehlung in das Nutzungskonzept aufgenommen.

Unter dem Aufgabenbereich *3.3 Anforderungen berücksichtigen* erprobten die Probanden zwei Anwendungsfälle. *Ideen zur Anforderungsberücksichtigung entwickeln* sowie *Potenzialanalyse anhand priorisierter Qualitätsmerkmale* wurde jeweils mithilfe von ChatGPT von einem Probanden erprobt und jeweils mit der Note 4 bewertet. Bei ersterem wurde der Chatbot als Inspirationsquelle für Umsetzungsmöglichkeiten konkreter Anforderungen verwendet. Bei letzterem wurden Technologien systematisch evaluiert, entsprechend ihrer Eignung bestimmte Anforderungen umzusetzen. Hierbei fiel dem Probanden das Wissen des Chatbots im Umfeld Java und Web-Technologien positiv auf. Da beide Anwendungsfälle von nur jeweils einem Probanden erprobt wurden, werden sie trotz ihrer guten Bewertung nicht als relevant eingestuft.

Unter *3.4 Lösungsarchitektur entwerfen* erprobten zwei Probanden mit Unterstützung von ChatGPT den Anwendungsfall *Komponentenschnitt erarbeiten*. Im Schnitt bewerteten sie mit $3,5$ und Standardabweichung von $0,5$. Hierbei wurde der Chatbot nach Vorschlägen zur Zerlegung eines Systems in Komponenten gefragt. Die Benotung dieses Anwendungsfalles wurde als herausfordernd wahrgenommen, da der Entwurf einer Lösungsarchitektur nicht immer richtig oder falsch sei, sondern differenziert abgewogen werden müsse. Trotzdem ist die Bewertung insgesamt positiv und stammt von mehr als einem Probanden, was diesen Anwendungsfall relevant macht. Anders als bei den anderen Anwendungsfällen aus dieser Domäne wurde der Chatbot hier nicht als Wissensressource, sondern als Inspirationsquelle für mögliche Zerlegungen verwendet. Die Unterstützung des Chatbots im Kontext individueller Situation und Anforderungen ist eine Neuerung, welche so von keinem herkömmlichen Werkzeuge geboten wird, das nicht auf LLMs basiert. Ein Nachteil dabei ist, dass eine textuelle Beschreibung dieses individuellen Kontexts (in Form eines Prompts) unter Umständen viel Zeit in Anspruch nehmen kann und den Geheimnisschutz verletzen kann. Obwohl ChatGPT kein echtes Verständnis von Software Architektur besitzt, hat das Werkzeug hier trotzdem Potenzial, das Software Engineering zu unterstützen. Auch dieser Anwendungsfall wird mit ChatGPT als Best Practice in das Nutzungskonzept aufgenommen.

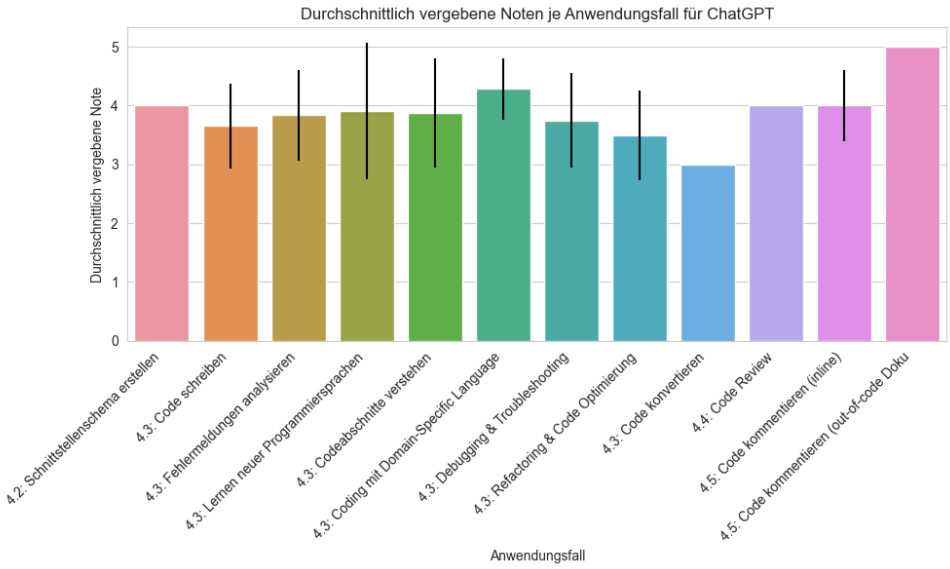
Ebenfalls unter *3.4 Lösungsarchitektur entwerfen* wurde *Trade-offs unterschiedlicher Lösungen vergleichen* von drei Probanden unter Verwendung von ChatGPT getestet und im Mittel mit $3,5$ (Standardabweichung $\approx 0,41$) benotet. Ähnlich zu den anderen Anwendungsfällen in dieser Domäne wurde der Chatbot dabei als Wissensressource verwendet, um Vor- und Nachteile bestimmter Lösungen zu beleuchten. Ein Hindernis dabei waren die sehr „diplomatischen" Antworten des Chatbots, der oft darauf hinwies „es komme drauf an", anstatt eine klare Position zu beziehen. Auch dieser Anwendungsfall ist entsprechend der oben definierten Metrik relevant und wird eingehender diskutiert. Über die Unterstützung als Wissensressource hinaus, kann der Chatbot auch hier auf eine individuelle Situation mit spezifischen Anforderungen eingehen. Das ist ein Feature, welches herkömmliche Werkzeuge nicht leisten können, und somit eine Neuerung für das Software Engineering. Zwar ist es wie in jedem Fall notwendig, ChatGPTs Ausgaben kritisch zu prüfen, doch hat der Chatbot hier das Potenzial einen Mehrwert zu bieten. Ähnlich zu dem vorangegangenen Anwendungsfall ist es auch hier notwendig, den individuellen Kontext in Form eines Prompts zu beschreiben, was viel Zeit in Anspruch nehmen kann und ebenfalls ein Risiko für den Geheimnisschutz sein kann. Dieser Anwendungsfall wird wird ebenfalls als Best Practice in das Nutzungskonzept aufgenommen.

Drei Anwendungsfälle erprobten die Probanden im Aufgabenbereich *3.5 Architektur dokumentieren und kommunizieren*. Mit ChatGPT und GitHub Copilot erprobte je ein Proband den Anwendungsfall *Architektur textuell dokumentieren*. Hierbei wurde die Architektur eines bestehenden Systems beschrieben. Beide Werkzeuge unterstützten bei der Formulierung der Texte, der Proband lieferte das Fachwissen. Sie bewerteten ihr Erfahrungen jeweils mit 4 (ChatGPT) und 3 (Copilot). Ein Proband erprobte mit ChatGPT den Anwendungsfall *UML-Diagramme erstellen* und bewertete seine Erfahrung mit Note 4. Auf Basis einer textuellen Beschreibung des Systems generierte der Chatbot Code in der PlantUML-Syntax, aus welches ein UML-Diagramm erzeugt werden kann. Kleine Änderungen am generierten UML Diagramm waren im Nachgang notwendig, um das gewünschte Ergebnis zu erhalten. *ER-Modelle zu SQL Create-Statements erstellen* wurde von einem Probanden mithilfe von ChatGPT erprobt und mit 3 benotet. Diese Funktionalität wurde durch den Einsatz von Plugins ermöglicht. Nach Erfahrung des Probanden funktionierten kleine Abschnitte gut, während größere Diagramme keine guten Ergebnisse lieferten. Keiner dieser drei Anwendungsfälle entspricht obiger Definition von Relevanz.

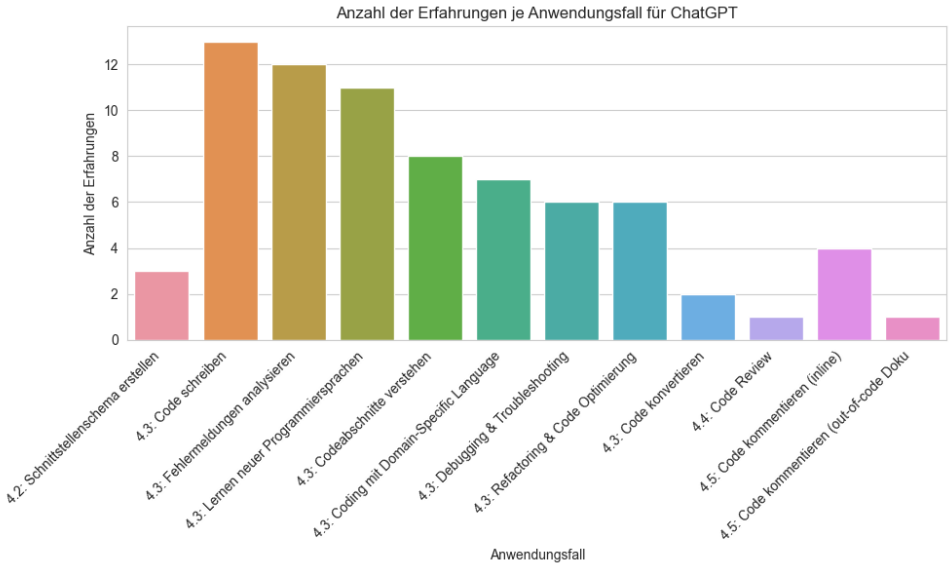
4.3.4 *Domäne 4. Implementierung*

Mit dreizehn Anwendungsfällen wurden in der vierten Domäne *Implementierung* die meisten Anwendungsfälle identifiziert.

Im Aufgabenbereich *4.2 Feindesign erstellen* erprobten drei Probanden unter Verwendung von ChatGPT den Anwendungsfall *Schnittstellenschema erstellen* jeweils mit Note 4. Hierbei geht es um die Erstellung von Schnittstellen-



(a) Durschnittliche Noten und Standardabweichung



(b) Anzahl der Erfahrungen

Abbildung 4.3: Im Mittel vergebene Noten der Probanden für die Anwendungsfälle in der Domänen 4 unter Verwendung von ChatGPT.

schemata (etwa im Format der JavaScript Object Notation (JSON)) auf Basis bereits vorhandener Datenformate. Dieser Anwendungsfall qualifiziert sich ebenfalls als relevant und wird näher besprochen. Diese Vermutung begründet sich nicht direkt auf den Ergebnissen der Untersuchungen — zu diesem Anwendungsfall wurden keine Freitextkommentare gemacht — doch liegt nahe, dass ChatGPT hier das Potenzial hat, zu einer großen Zeitersparnis beizutragen. Zum einen ist das Generieren von technik-nahem Code sowie Boilerplate Code eine große Stärke des Chatbots, zum anderen fällt ein Hindernis für den Effizienzgewinn weg: Die an den Chatbot zu übermittelnden Kontextinformationen sind bereits in Form der Datenformate beschrieben. Dadurch beschleunigt sich die Formulierung eines Prompts enorm. Es ist lediglich notwendig, das gewünschte Schema-Format, die bestehenden Datenformate sowie die Constraints anzugeben. Eine Voraussetzung für den Einsatz von ChatGPT ist jedoch, dass der Geheimnisschutz gewahrt bleibt. Die Übermittlung der bereits bestehenden Datenformate könnte sensible Informationen beinhalten. Alles in allem wird dieser Anwendungsfall als Best Practice und klare Empfehlung in das Nutzungskonzept aufgenommen.

Das Aufgabenbereich *4.3 Code schreiben* umfasst mit 9 die meisten identifizierten Anwendungsfälle. Den gleichnamigen Anwendungsfall *Code schreiben* erprobten 13 Probanden mit ChatGPT und 11 Probanden mit GitHub Copilot. Durchschnittliche Noten und Standardabweichung waren $3,65$ und $\approx 0,72$ für die Arbeit mit ChatGPT und $3,41$ bzw. $\approx 0,67$ mit Copilot. Dabei unterstützen die Werkzeuge auf unterschiedliche Weise. ChatGPT muss via Prompts explizit gefragt werden Code zu generieren, während Copilot in der Entwicklungsumgebung integriert laufend Vorschläge unterbreitet. Der von dem Chatbot generierte Code wies nach den Freitextkommentaren der Probanden teilweise Fehler und Mängel auf, oder ging an den textuell beschriebenen Anforderungen vorbei. Diese Fehler konnten teilweise vom Chatbot selbst korrigiert werden, nachdem dieser dazu aufgefordert wurde. Des Weiteren fanden die Probanden die Unterstützung durch ChatGPT im Kontext eines neuen „grüne Wiese“ Projektes hilfreicher als im Kontext von großen bestehenden Systemen. Die Unterstützung von GitHub Copilot stand ebenfalls vor Herausforderungen. Je technik-ferner und fachlich komplexer die zu implementierende Funktionalität, desto weniger hilfreich schienen die Vorschläge von Copilot. Auch bei diesem Werkzeug war der generierte Code fehleranfällig, musste hier allerdings händisch korrigiert werden. Trotz dieser Herausforderungen handelt es sich für beide Werkzeuge um einen relevanten Anwendungsfall, der näher diskutiert wird. In beiden Fällen ist die Standardabweichung höher als der nach-Komma-Anteil der durchschnittlichen Bewertung, sodass die Bewertungen im Mittel auch geringer als „durchwachsen“ ausgefallen sein können. Zwar ist eine hohe Standardabweichung bei der Vielzahl an Erfahrungen für diesen Anwendungsfall nicht überraschend, dennoch fällt die Bewertung der Probanden hier nicht durchweg positiv aus. Die Freitextantworten zu den Fragen 6 und 7 (siehe Unterabschnitt 4.2.3) erwähnen den Anwendungsfall „Code schreiben“ häufig. Demnach scheint ChatGPT besonders im Umgang mit unbekannten Programmiersprachen und im Falle von

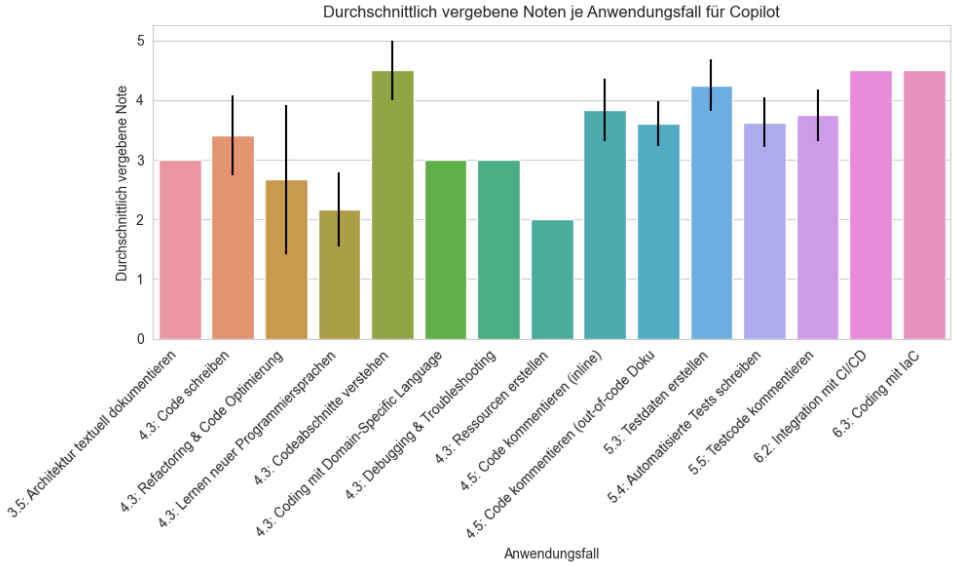
Boilerplate Code und dem Erreichen einer ersten, lauffähigen Version zu unterstützen. Copilot hätte hingegen einen neutralen bis negativen Effekt auf die Codequalität, insofern diese nicht laufen geprüft und korrigiert wird, unterstützt aber sehr bei der Implementierung von repetitiven Codeabschnitten — obgleich dieser Zeitgewinn durch die Überprüfung der Ausgaben zurückgehalten wird. Darüber hinaus könne Copilot erst dann effektiv helfen, wenn die ersten Schritte in der Implementierung bereits gemacht wurden. Die Vermutung liegt nahe, die Werkzeuge könnten sich gut ergänzen: ChatGPT könnte auf Basis einer textuellen Beschreibung einen ersten Entwurf generieren, der dann mithilfe von Copilot weiter bearbeitet werden könnte. Ob diese Vermutung in der Praxis zutrifft, lässt sich auf Basis der Untersuchungen nicht sagen. Es scheint, als wäre der Mehrwert beider Werkzeuge stark unterschiedlich und von einigen Parametern abhängig — etwa von dem Projektkontext, der Erfahrung des Nutzers, den verwendeten Technologien und der individuellen Arbeitsweise. Alles in allem hat die Unterstützung der Werkzeuge in diesem Anwendungsfall für einige Probanden einen Mehrwert dargestellt. Deshalb wird der Anwendungsfall für beide Werkzeuge als Best Practice in das Nutzungskonzept aufgenommen.

Der Anwendungsfall *Refactoring & Code Optimierung* liegt ebenfalls im Aufgabenbereich *4.3 Code schreiben* und wurde von 6 Probanden mit Unterstützung durch ChatGPT und 3 Probanden mit Unterstützung von GitHub Copilot getestet. Die Unterstützung von ChatGPT wurde im Schnitt mit $3,5$ bewertet (Standardabweichung $\approx 0,76$), die von Copilot im Schnitt mit $2,67$ bei einer Standardabweichung von $\approx 1,25$. Auch hier unterstützen die Werkzeuge auf sehr unterschiedliche Art. Während ChatGPT aktiv nach Anpassungsvorschlägen gefragt wurde, blieb Copilot schlicht aktiv, während die Probanden Refactorings durchführten. Der Assistent lässt sich nicht für einen Anwendungsfall speziell konfigurieren, sondern arbeitet stets im selben Modus. Bei diesem Anwendungsfall scheint die unterschiedliche Art der Unterstützung einen großen Einfluss auf die Erfahrung der Probanden zu haben. Während die Hilfe des Chatbots zwischen „durchwachsen“ und „erfolgreich“ bewertet wurde, ist die Erfahrung mit Copilot im Mittel schlechter als „durchwachsen“ bei einer sehr hohen Standardabweichung bewertet worden. Da die Unterstützung von ChatGPT zudem von mehr als einem Probanden erprobt wurde, ist dieser Anwendungsfall mithilfe des Chatbots als relevant eingestuft und wird weiter diskutiert. Es ist denkbar, dass Copilot ungeeignet für diesen Anwendungsfall ist, da der Assistent nicht darauf ausgelegt ist, dabei zu unterstützen. Eine weitere Einschränkung ist der Geheimnisschutz. Bei bestehendem, zu optimierenden Code könnte es sich um sensible Informationen handeln. Die hohe Standardabweichung — auch bei ChatGPT — lässt sich womöglich darauf zurückführen, dass der Mehrwert der Unterstützung sehr auf die individuelle Situation ankommt, wie im vorangegangenen Anwendungsfall diskutiert. Insgesamt scheint hier Potenzial für den Einsatz von ChatGPT zu bestehen. Auch dieser Anwendungsfall wird unter Verwendung von ChatGPT als Best Practice in das Nutzungskonzept aufgenommen.

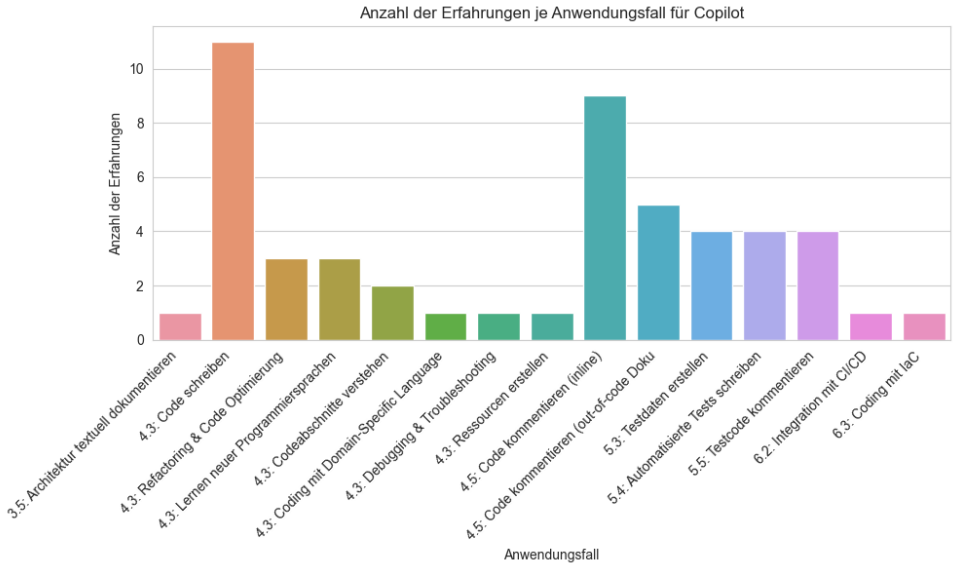
Der Anwendungsfall *Debugging & Troubleshooting* liegt auch im Aufgabenbereich *4.3 Code schreiben*, wurde von 6 Probanden mit ChatGPT erprobt und durchschnittlich mit 3,75 benotet. Die Standardabweichung liegt bei $\approx 0,80$. Es geht hierbei um das Identifizieren und Beheben von Fehler und deren Ursachen in einem System. Im Fall von ChatGPT wurden Codeabschnitte an den Chatbot gegeben und nach möglichen Fehlerursachen gefragt. Den Freitextkommentaren zufolge sei der Chatbot gut darin, in die grobe Richtung einer korrekten Antwort zu weisen, werde aber durch sein zeitlich begrenztes Wissen zurückgehalten. Darüber hinaus erprobte ein Proband den Anwendungsfall GitHub Copilot und benotete seine Erfahrung mit 3. Copilot habe nicht dabei geholfen, den Fehler zu verstehen, jedoch erleichterte das Werkzeug das Schreiben repetitiver „Print Debugging Statements“. Mit ChatGPT erfüllt dieser Anwendungsfall die Kriterien für Relevanz und wird deshalb näher besprochen. Während die Bewertung der Erfahrung der sechs Probanden in Richtung „erfolgreich“ tendiert, ist die Standardabweichung relativ hoch. Ob das an ChatGPTs fehleranfälligen Ausgaben, oder an Unterschieden in der jeweiligen Situation der Probanden liegt, lässt sich auf Basis der Untersuchungen nicht sagen. Auch wird der Anwendungsfall durch die zeitlich begrenzten Trainingsdaten des Chatbots zurückgehalten. Dadurch könnte er unter Umständen nicht über die notwendigen Informationen verfügen, um bei aktuellen oder neuen Fehlern zu helfen. Trotzdem wird auch dieser Anwendungsfall unter Verwendung von ChatGPT als Best Practice in das Nutzungskonzept aufgenommen, mit der Empfehlung sie selbst zu erproben und den Mehrwert in der eigenen Situation zu bewerten.

Ein weiterer unter *4.3 Code schreiben* identifizierter Anwendungsfall, ist *Fehlermeldungen analysieren*. Dieser wurde von 12 Probanden unter Verwendung von ChatGPT erprobt, im Schnitt mit 3,83 bei einer Standardabweichung von $\approx 0,77$ bewertet. Der Anwendungsfall ist dem vorherigen „Debugging & Troubleshooting“ sehr ähnlich. Hier geht es explizit darum, Fehlermeldungen auf den Grund zu gehen. Er wurde aufgrund häufiger expliziter Nennungen sowie um die Erfahrungen der Probanden möglichst genau abzubilden trotz seiner Ähnlichkeit zum vorherigen Anwendungsfall aufgenommen. Auch hier half ChatGPT dabei in die Richtung einer korrekten Antwort zu weisen, weniger aber beim Beheben der Ursache. Teilweise nahmen die Probanden die Antworten des Chatbots als zu allgemein wahr. Auch dieser Anwendungsfall entspricht der oberen Definition von Relevanz und wird näher erörtert. ChatGPT bietet hier eine Alternative zu herkömmlichen Quellen wie etwa der Websuche oder Plattformen wie Stack Overflow. Was den Chatbot dabei auszeichnet, ist seine Schnelligkeit gegenüber diesen Quellen, da ein Nutzer im Umgang mit dem Chatbot keine große Menge an Informationen sichten und durchsuchen muss. Auch wenn der Chatbot aufgrund seiner zeitlich begrenzten Trainingsdaten und Fehleranfälligkeit unter Umständen weniger verlässlich als eine Websuche ist. Der Geschwindigkeitsvorteil wird durch die Freitextantworten auf Frage 7 bestätigt. Ein weiterer Vorteil ist die Leichtgewichtigkeit dieses Anwendungsfalles: So kann der Anwendungsfall ohne viel Aufwand erprobt und schnell abgebrochen werden, sollte der ChatGPT hier kei-

nen Mehrwert liefern. Zu Unterstreichen ist auch die Vielzahl an Probanden, welche diesen Anwendungsfall als tendenziell „erfolgreich“ bewertet haben. Der Anwendungsfall wird als Best Practice wird in das Nutzungskonzept aufgenommen.



(a) Durschnittliche Noten und Standardabweichung



(b) Anzahl der Erfahrungen

Abbildung 4.4: Im Mittel vergebene Noten der Probanden für die Anwendungsfälle in allen Domänen unter Verwendung von Copilot.

Auch unter *4.3 Code schreiben* wurde der Anwendungsfall *Codeabschnitte verstehen* von 8 Probanden unter Verwendung von ChatGPT erprobt. Im Mittel vergaben sie die Note $3,88$ bei einer Standardabweichung von $\approx 0,93$. Unter Verwendung von GitHub Copilot waren es 2 Probanden, die ihre Erfahrung mit der Note $4,5$ (Standardabweichung $0,5$) bewerteten. Während ChatGPT

via Prompt explizit nach Erklärungen zu Codeabschnitten gefragt wurde, ließen die Probanden Copilot Abschnitte erklären, indem sie oberhalb der entsprechenden Codezeilen einen Kommentar eröffneten und den Assistenten dessen Inhalt generieren ließen. In beiden Fällen fällt die Bewertung sehr positiv aus und qualifiziert diesen Anwendungsfall als relevant. Die hohe Standardabweichung bei ChatGPT lässt wieder darauf schließen, dass zumindest einige der Probanden einen Mehrwert in diesem Anwendungsfall fanden. Bei Copilot könnte die Erfahrung besser sein, weil der eigene Workflow nicht verlassen werden muss, um die Erklärung zu bekommen. Auch ein Faktor könnten Parallelen zwischen diesem und dem Anwendungsfall „Code kommentieren (inline)“ sein, welcher weiter unten diskutiert wird. Auch dort wurde die Erfahrung mit Copilot als positiv wahrgenommen. In beiden Fällen ist die Einhaltung des Geheimnisschutzes auch ein Aspekt, der potenzielle Nutzung einschränken könnte. Alles in allem ist eine Evaluation dieses Anwendungsfalles schwer — auch da keine Freitextkommentare zu diesem abgegeben wurden. Trotzdem wird der Anwendungsfall für beide Werkzeuge als Best Practice in das Nutzungskonzept aufgenommen. Ausschlaggebend dafür sind die „erfolgreiche“ bis „deutlich erfolgreiche“ Bewertung.

Ebenfalls unter dem Aufgabenbereich *4.3 Code schreiben* erprobten 11 Probanden mit ChatGPT und 3 mit GitHub Copilot den Anwendungsfall *Lernen neuer Programmiersprachen*. Die Unterstützung von ChatGPT bewerteten die Probanden durchschnittlich mit $3,91$ bei einer Standardabweichung von $\approx 1,16$. Der Chatbot unterstützte beim Lernen neuer Aspekte in bereits vertrauten Programmiersprachen sowie beim Aneignen Neuer. Als Hindernis wurde wahrgenommen, dass der generierte Code nicht zwangsweise mit einem Lerneffekt einherging. Die Unterstützung von Copilot wurde im Schnitt mit $2,17$ (Standardabweichung $\approx 0,62$) bewertet. Ähnlich wie bei dem Anwendungsfall „Refactoring & Code Optimierung“ gibt es für Copilot hier keine besonderen Einstellungen, welche den Assistenten auf einen Anwendungsfall konfigurieren. Zwar wurden die Vorschläge des Werkzeugs teilweise als dem Lernprozess zuträglich wahrgenommen, doch gehören Erklärungen nicht zum Featureumfang von Copilot. Nur mit Unterstützung durch ChatGPT wird der Anwendungsfall als relevant eingestuft und näher diskutiert. In den Freitextantworten zu Frage 6 erwähnten drei Probanden, die Unterstützung von ChatGPT hätte ihnen beim Lernen einer neuen Programmiersprache geholfen. Dabei hätte der Chatbot auch Fragen beantworten können, welche sie sonst an Kollegen hätten stellen müssen. Auch die Unterstützung in den oben genannten Anwendungsfällen „Debugging & Troubleshooting“, „Fehlermeldungen analysieren“ und „Codeabschnitte verstehen“ kann beim Lernen neuer Programmiersprachen helfen. Und während die Note tendenziell „erfolgreich“ ist, gehen die Meinungen der Probanden stark auseinander. Das könnte wie bei den anderen Fällen auch an einem hohen Einfluss der individuellen Situation, oder aber an der Fehleranfälligkeit des Chatbots liegen. Dennoch wird der Anwendungsfall als Best Practice in das Nutzungskonzept aufgenommen, da einige Probanden sehr positive Erfahrungen gemacht haben.

Ein weiterer Anwendungsfall unter *4.3 Code schreiben* ist *Coding mit Domain-Specific Languages*. Dieser ist eine Spezialisierung von „Code schreiben“ und wurde aufgrund mehrfacher expliziter Nennungen aufgenommen, um die Erfahrung der Probanden möglichst genau abzubilden. Hierbei geht es um das Programmieren mit sogenannten domänenspezifischen Sprachen (DSL): Spezialisierte Programmiersprachen, von begrenzter Ausdrucksmächtigkeit für den Einsatz in spezifischen Domänen [Fow10, S. 27]. Dazu zählen beispielsweise die Structured Query Language (SQL) oder reguläre Ausdrücke. Der Anwendungsfall wurde unter Verwendung von ChatGPT's Unterstützung durch 7 Probanden erprobt und im Schnitt mit 4, 29 bei einer Standardabweichung von 0, 52 bewertet. Der Chatbot beantwortete Fragen zu Konzepten, Syntax und generiertem Code. Ein Proband erprobte den Anwendungsfall mit GitHub Copilot und bewertete seine Erfahrung mit 3. Im Fall von ChatGPT ist dieser Anwendungsfall relevant. Hier scheint die Unterstützung des Werkzeugs einen großen Mehrwert zu bieten. Im Mittel liegt die Bewertung zwischen „erfolgreich“ und „deutlich erfolgreich“ und ist damit signifikant höher als bei dem allgemeineren Anwendungsfall „Code schreiben“. Die dazu relativ niedrige Standardabweichung lässt darauf schließen, dass die meisten Probanden diesen positiven Effekt wahrgenommen haben. Es erscheint denkbar, dass die Kompetenzen des Chatbots in der Unterstützung bei den anderen Anwendungsfällen im Aufgabenbereich „Code schreiben“ auch auf DSLs übertragbar sind. Neben den üblichen Nachteilen von ChatGPT — potenziell zeitaufwendiges Zusammentragen der Kontextinformationen, Achtung auf den Geheimnisschutz — ist eine weitere Einschränkung dieses Anwendungsfalles, dass ein komplexes Ergebnis unter Umständen nur schwer von den Engineers nachvollzogen und überprüft werden kann. Alles in allem wird der Anwendungsfall unter Verwendung von ChatGPT als Best Practice und klare Empfehlung in das Nutzungskonzept aufgenommen.

Auch unter dem Aufgabenbereich *4.3 Code schreiben* wurde der Anwendungsfall *Ressourcen erstellen* von einem Probanden unter Verwendung von GitHub Copilot getestet und mit 2 bewertet. Es ging dabei um die Erstellung einer Scalable Vector Graphics (SVG) Datei, welche als Platzhalter dienen sollte. Dieser Anwendungsfall genügt nicht der oberen Definition von Relevanz und wird deshalb nicht weiter diskutiert.

Der letzte unter *4.3 Code schreiben* identifizierte Anwendungsfall ist *Code konvertieren*. Dabei geht es um die Konvertierung von Code zwischen unterschiedlichen Programmiersprachen, Frameworks oder Versionen. Der Anwendungsfall wurde von 2 Probanden mit Unterstützung von ChatGPT erprobt. Beide bewerteten ihre Erfahrung mit 3. Das macht auch diesen Anwendungsfall nicht relevant.

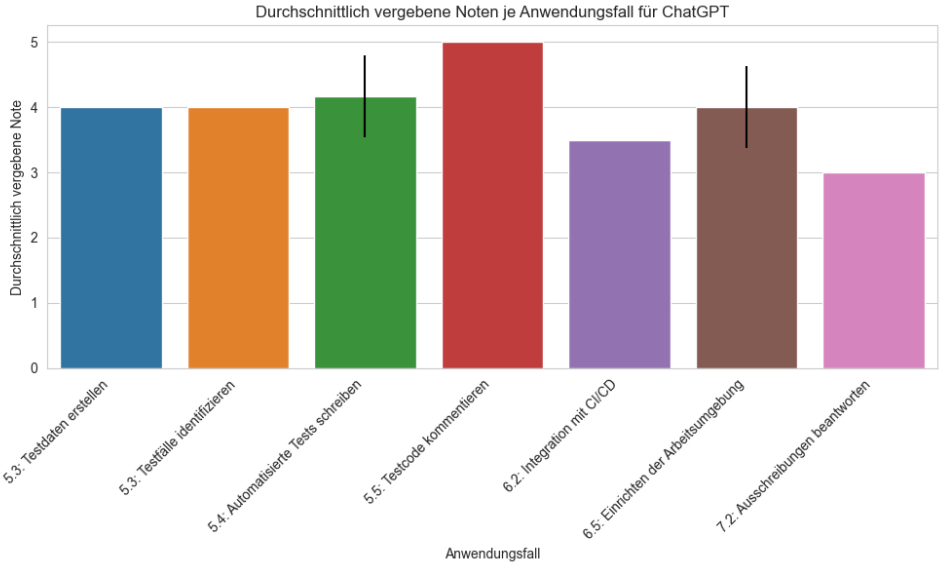
Unter dem Aufgabenbereich *4.4 Codequalität absichern* wurde der Anwendungsfall *Code Review* identifiziert und von einem Probanden unter der Verwendung von ChatGPT erprobt. Er bewertete seine Erfahrung mit der Note 4. Konkret ging es darum, die Qualität des eigenen Code von dem Chatbot bewerten zu lassen. Trotz der „erfolgreichen“ Bewertung wird dieser Anwen-

dungsfall nicht als relevant eingestuft, da es sich um eine Einzelerfahrung handelt.

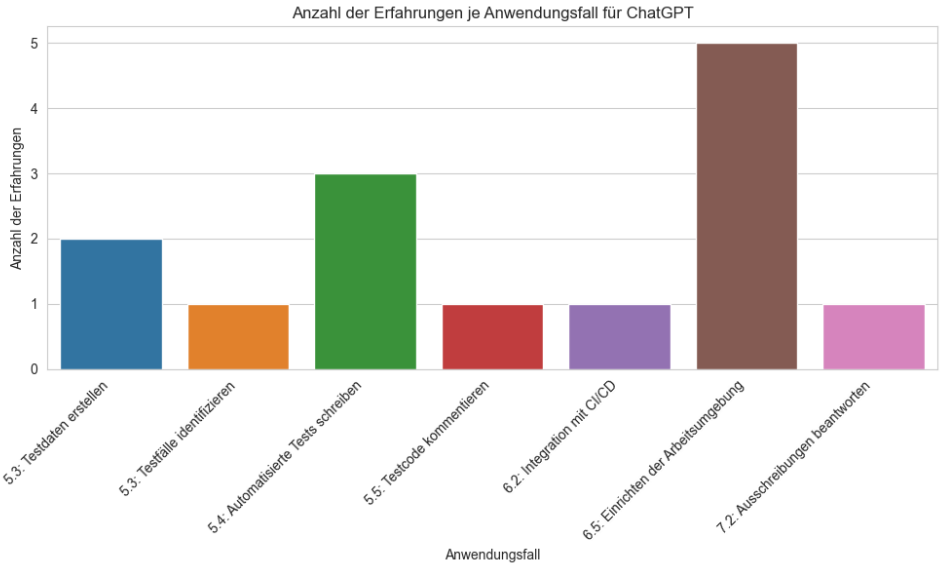
Im Aufgabenbereich *4.5 Code dokumentieren* wurden zwei Anwendungsfälle identifiziert, welche einander sehr ähnlich sind: *Code kommentieren* (*inline*) und *Code kommentieren* (*out-of-code Dokumentation*). „Inline" bezeichnet hier Kommentare im Quellcode, während „out-of-code" Dokumentation separat vom Quellcode liegt. Ersteren Anwendungsfall erprobten 4 Probanden mit ChatGPT (durchschnittliche Note $4,0$, Standardabweichung $\approx 0,61$) und 9 Probanden mit GitHub Copilot (durchschnittliche Note $3,83$, Standardabweichung $\approx 0,53$). ChatGPT erstellte Kommentare als Teil des generierten Codes, benötigte hierzu allerdings eine detaillierte Beschreibung des gewünschten Ergebnisses. Copilot unterbreitete häufig sinnvolle Vorschläge, deren Qualität jedoch stark von der Komplexität des zu kommentierenden Codes abhing. Je komplexer der Code oder je mehr Fachlichkeit er enthielt, desto schlechter die Vorschläge. Eine manuelle Korrektur der Vorschläge war teilweise erforderlich. In beiden Fällen handelt es ich um einen relevanten Anwendungsfall, der weitergehend besprochen wird. Auch wenn ChatGPT im Mittel besser bewertet wurde, bestehen zwei Einschränkungen. Zum einen muss der eigene Workflow verlassen werden, um den Chatbot aktiv um die Generierung von Kommentaren zu bitten. Dabei kann es notwendig sein, Kontextinformationen aufzubereiten. Zum anderen erprobten die Probanden lediglich die Generierung von Kommentaren als Teil der Generierung von Code. Inwiefern ChatGPT sich dazu eignet, Kommentare zu bereits bestehendem Code zu generieren, lässt sich auf Basis der Untersuchungen nicht zweifelsfrei ableiten — obgleich dieser Schluss naheliegt. Die Unterstützung von Copilot wurde unterdes im Schnitt zwar im Vergleich weniger gut bewertet, doch wurde dieser Anwendungsfall in den Freitextkommentaren zu Frage 6 und 7 von drei beziehungsweise fünf Probanden positiv erwähnt. Demnach sei die Qualität der mithilfe des Assistenten geschriebenen Kommentare und Dokumentation besser und es stelle sich ein Geschwindigkeitszuwachs ein. Alles eingeschlossen wird der Anwendungsfall für beide Werkzeuge als Best Practice in das Nutzungskonzept aufgenommen, wobei die Unterstützung durch Copilot sogar eine klare Empfehlung ist.

Zweiteren Anwendungsfall unter *4.5 Code dokumentieren*, *Code kommentieren* (*out-of-code Dokumentation*), erprobte 1 Proband mit ChatGPT (Note 5) und 5 Probanden mit GitHub Copilot. Durchschnittliche Bewertung und Standardabweichung bei Copilot waren $3,6$ und $\approx 0,37$. Auch hier waren die Vorschläge Copilot grundsätzlich sinnvoll, gingen jedoch nicht selten in die falsche Richtung. Es entstand der Eindruck, das Verständnis vom Kontext fehle. Trotz der Bestnote für ChatGPT, entspricht nur die Erfahrung mit Copilot der obigen Definition von Relevanz. Wie im vorherigen Absatz erwähnt, wurde die Unterstützung durch den Assistenten bei der Kommentierung als auch der Dokumentation von Code sehr positiv erwähnt. Ob die minimal geringere Bewertung von „out-of-code" gegenüber „inline" tiefer begründbar ist, ist schwer zu sagen. Diese Differenz könnte jedoch an der Schwäche von Copilot bei höherer fachlicher Komplexität und durch höhere Distanz zum Code lie-

gen. Ein Vorteil von Copilot in diesem Kontext ist auch, dass die Vorschläge direkt in der Entwicklungsumgebung unterbreitet werden, und ignoriert werden können, sollten sie unpassend sein. Insgesamt wird dieser Anwendungsfall als Best Practice in das Nutzungskonzept aufgenommen.



(a) Durschnittliche Noten und Standardabweichung



(b) Anzahl der Erfahrungen

Abbildung 4.5: Im Mittel vergebene Noten der Probanden für die Anwendungsfälle in den Domänen 5, 6 und 7 unter Verwendung von ChatGPT.

4.3.5 *Domäne 5. Test*

In der fünften Domäne *Test* erprobten die Probanden vier Anwendungsfälle.

Unter Aufgabenbereich *5.3 Manuell testen* wurde der Anwendungsfall *Testdaten erstellen* von 2 Probanden unter Verwendung von ChatGPT (beide vergaben Note 4) und 4 Probanden mit GitHub Copilot (durchschnittliche Note 4, 25; Standardabweichung $\approx 0,43$) erprobt. Im Falle von Copilot erzeugte das Werkzeug die Testdaten als Teil des generierten Codes. Beide Anwendungsfälle wurden von mehr als einem Probanden erprobt und sogar mit „erfolgreich" oder besser bewertet. Das macht beide relevant, sodass sie hier weiter diskutiert werden. Bei ChatGPT wurden nur wenige Freitextkommentare gegeben, sodass eine Evaluation hier schwerfällt. Für Copilot könnte es zu einem hohen Geschwindigkeitsgewinn beitragen, dass die Testdaten direkt als Teil des generierten Codes erstellt werden. In beiden Fällen ist jedoch Vorsicht geboten: Maschinelles Lernen beinhaltet immer Bias, sodass die erstellten Testdaten unter Umständen nicht repräsentativ sind und zu Klischees oder Allgemeinem tendieren, wie in Unterabschnitt 2.4.3 beschrieben. Gerade die Tendenz zum Allgemeinen könnte dazu führen, dass Edge-Cases nicht abgedeckt werden. Trotz dieser Einschränkung wird der Anwendungsfall für beide Werkzeuge als Best Practice in das Nutzungskonzept aufgenommen, da großes Potenzial für Zeitersparnis besteht. Für Copilot handelt es sich sogar um eine klare Empfehlung, da die Unterstützung des Werkzeuges von vielen Probanden sehr positiv wahrgenommen wurde.

Ebenfalls unter *5.3 Manuell testen* wurde der Anwendungsfall *Testfälle identifizieren* von einem Probanden mithilfe von ChatGPT erprobt und mit 4 bewertet. Da es sich um eine Einzelerfahrung handelt, fällt dieser Anwendungsfall nicht unter die Definition von Relevanz und wird somit nicht näher betrachtet.

Unter dem Aufgabenbereich *5.4 Automatisiert testen* erprobten 3 Probanden mit ChatGPT und 4 Probanden mit GitHub Copilot den Anwendungsfall *Automatisierte Tests schreiben*. Die Unterstützung durch ChatGPT wurde im Mittel mit 4, 17 bei einer Standardabweichung von $\approx 0,62$ bewertet. Dabei wurde der Chatbot gebeten, Testcode für bestehenden Code zu schreiben. Copilots Unterstützung bewerteten die Probanden im Mittel mit 3, 62 bei einer Standardabweichung von $\approx 0,41$. Hier unterstütze Copilot ähnlich wie bei dem Anwendungsfall „Code schreiben". Beide sind relevant und werden weiter diskutiert. Im Fall von ChatGPT könnte viel Zeit durch die Generierung des Testcodes gespart werden, gerade da der zu testende Code (und damit ein relevanter Teil des zu übermittelnden Kontextes) bereits vorliegt. Auch Copilot scheint für diesen Anwendungsfall gut geeignet zu sein, da Testcode oft sehr repetitiv ist. So wurde die Unterstützung des Assistenten im Vergleich zum Anwendungsfall „Code schreiben" hier besser bewertet. Da in beiden Fällen der genaue Umfang des Testcodes in den Untersuchungen nicht erfasst wurde, fällt ein Fazit hier schwer. Aufgrund des Potenzials beider Werkzeuge in diesem Anwendungsbereich wird auch dieser Anwendungsfall als Best Practice in das Nutzungskonzept aufgenommen.

Unter *5.5 Tests dokumentieren und auswerten* erprobte ein Proband den Anwendungsfall *Testcode kommentieren* mit ChatGPT und bewertete seine Erfahrung mit 5. Mit GitHub Copilot waren es 4 Probanden, die ihre Erfahrung

durchschnittlich mit 3,75 bei einer Standardabweichung von $\approx$ 0,43 bewerteten. In beiden Fällen wurden die Kommentare als Teil des generierten Testcodes erstellt. Bei der Arbeit mit Copilot gab ein Proband an, das hohe Maß an Fachlichkeit im Vergleich zu „normalem" Code sei eine Herausforderung für das Werkzeug gesehen. Trotz der hohen Bewertung von ChatGPT, ist der Anwendungsfall im Rahmen dieser Auswertung nur für Copilot relevant. Obgleich der Herausforderung durch fachlich komplexeren Code wurde Copilots Unterstützung hier im Mittel nur minimal schlechter bewertet als bei dem Anwendungsfall „Code kommentieren (inline)". Womöglich liegt der Grund hierfür in der repetitiven Natur von Testcode, wie im vorherigen Anwendungsfall diskutiert. Gemeinsam mit den sehr positiven Berichten aus den Freitextantworten zu Fragen 6 und 7 bezüglich Copilots Unterstützung beim Kommentieren von Code wird auch dieser Anwendungsfall als Best Practice in das Nutzungskonzept aufgenommen.

4.3.6 *Domäne 6. Infrastruktur & Betrieb*

In der sechsten Domäne *Infrastruktur & Betrieb* wurden im Rahmen der Untersuchungen drei Anwendungsfälle identifiziert.

Unter Aufgabenbereich *6.2 Infrastruktur entwickeln* erprobte je ein Proband den Anwendungsfall *Integration mit CI/CD* mit ChatGPT (Note 3,5) und GitHub Copilot (Note 4,5). Dabei ging es um die Integration eines bestehenden Projektes in DevOps-Abläufe. Dazu wurden die Dateien zur Konfiguration dieser Abläufe mithilfe der Werkzeuge erstellt und bearbeitet. In beiden Fällen handelt es sich um Einzelerfahrungen, sodass dieser Anwendungsfall als nicht relevant eingestuft wird.

Unter *6.3 Prozesse automatisieren* sammelte ein Proband Erfahrung im Anwendungsfall *Coding mit IaC* mit GitHub Copilot und bewertete seine Erfahrung mit 4,5. Ähnlich wie beim vorherigen Anwendungsfall wurden Dateien zur Konfiguration — in diesem Fall zur Konfiguration der Infrastruktur — mithilfe der Werkzeuge erstellt und bearbeitet. Auch hierbei handelt es sich um eine Einzelanwendung und somit um einen nicht relevanten Anwendungsfall, der nicht weiter besprochen wird.

Im Aufgabenbereich *6.5 Entwicklungsumgebung bereitstellen* erprobten 5 Probanden mithilfe von ChatGPT den Anwendungsfall *Einrichten der Arbeitsumgebung*. Sie bewerteten ihre Erfahrung im Mittel mit 4, bei einer Standardabweichung $\approx$ 0,63. Bei diesem Anwendungsfall geht es um die Einrichtung und Konfiguration einer Arbeitsumgebung, um die Arbeit an einem Projekt zu ermöglichen. Hierbei beantwortete der Chatbot Fragen zu Konfigurationen und Einstellungen. Der Anwendungsfall erfüllt die Kriterien für Relevanz und wird deshalb weiter diskutiert. Auch hier wurden keine Freitextantworten gegeben, sodass eine Evaluation schwerfällt. Ähnlich wie bei dem Anwendungsfall „Fehlermeldungen analysieren" könnte der Chatbot hier eine Alternative zu herkömmlichen Quellen wie etwa der Websuche darstellen. Das kommt mit denselben Vor- und Nachteilen: Ein Nutzer ist nicht gezwungen eine potenziell große Menge an Informationen zu sichten und zu bewer-

ten, jedoch ist der Chatbot aufgrund seiner zeitlich begrenzten Trainingsdaten und Fehleranfälligkeit unter Umständen weniger verlässlich als eine Websuche. Auch dieser Anwendungsfall ist ähnlich leichtgewichtig, kann ohne viel Aufwand erprobt und schnell abgebrochen werden, insofern kein Mehrwert entsteht. Der Geheimnisschutz scheint hier ebenfalls weniger problematisch, da die Einrichtung der Umgebung in der Regel nicht sensibel ist. Insgesamt wird dieser Anwendungsfall auch aufgrund seiner im Mittel „erfolgreichen" Bewertung als Best Practice und klare Empfehlung in das Nutzungskonzept aufgenommen.

4.3.7 *Domäne 7. Projektmanagement*

In der siebten Domäne *Projektmanagement* wurde lediglich ein Anwendungsfall unter dem Aufgabenbereich *7.2 Projekt planen und kalkulieren* identifiziert: *Ausschreibungen beantworten*. Dieser wurde von einem Proband mit ChatGPT erprobt und mit der Note 3 bewertet. Eine Ausschreibung bezeichnet in diesem Kontext eine Aufforderung, Angebote für die in der Ausschreibung genannten Leistungen abzugeben [Wik23a]. Ein Freitextkommentar blieb aus. Dieser Anwendungsfall wird als nicht relevant eingeschätzt und nicht weiter diskutiert.

4.4 AUSWERTUNG UND SCHLUSSFOLGERUNGEN

Dieser Abschnitt präsentiert eine Auswertung der Untersuchungsergebnisse. Über die im Fragebogen hinaus erfassten Einschätzungen der Probanden zu Qualität und Effizienz ihrer Arbeit in Unterstützung durch die Werkzeuge, geht es hier um die Identifizierung weiterer Faktoren, die die Arbeit mit den Werkzeugen beeinflussen. Dazu werden über die nachfolgenden Unterabschnitte Zusammenhänge zwischen den erfassten Variablen untersucht, Unterschiede in der verwendeten Variante von ChatGPT verglichen und die von den Probanden angegebenen potenziale der Werkzeuge für die Zukunft diskutiert. Es werden Rückfragen an die Probanden besprochen, auf Kritik und Limitationen der Untersuchung eingegangen und ein Fazit über die Untersuchungen gezogen.

4.4.1 *Zusammenhänge zwischen Kontext und Antworten*

Dieser Unterabschnitt untersucht mögliche Zusammenhänge zwischen dem, durch Fragen 1 bis 3 erhobenen, Kontext und den Antworten auf die einordnenden Fragen 6 *Qualität*, 7 *Effizienz* und 8 *Fähigkeiten/Lerneffekt*. Ziel ist dabei die Identifizierung von Unterschieden in den Antworten, welche auf Parametern des Kontexts zurückzuführen sind. Ein Beispiel für eine solche Abhängigkeit ist etwa die Hypothese:

> *„Die weniger erfahrenen Software Engineers erleben durch die Unterstützung von ChatGPT einen größeren Lerneffekt als erfahrenere Software*

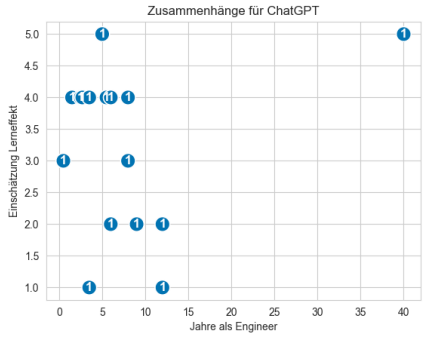
Engineers.“ — Eine unbestätigte Hypothese, welche lediglich der Illustration dient.

Um derartige Abhängigkeiten zu untersuchen, werden jeweils einzelne Parameter des Kontexts auf die Antworten zu einer der Fragen 6 bis 8 abgebildet. Diese Untersuchung geschieht lediglich visuell in der Form von Streudiagrammen. Es wurde absichtlich auf die Verwendung statistischer Algorithmen Methoden verzichtet, da diese aufgrund der geringen Datenmenge von 16 Probanden mit ChatGPT bzw. 11 Probanden im GitHub Copilot keine aussagekräftigen Ergebnisse liefern würden. Abbildung 4.6 zeigt eine Gruppe solcher Diagramme. Die folgenden Parameter des Kontexts werden dabei je KI-Werkzeug auf der x-Achse abgebildet:

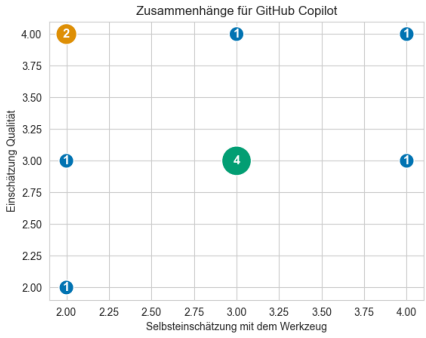
- Die Anzahl der Jahre, die ein Proband bereits als Software Engineer arbeitet.
- Die zeitliche Erfahrung im Umgang mit dem jeweiligen Werkzeug zum Zeitpunkt des Interviews. Wie in Unterabschnitt 4.2.1 dargelegt, handelt es ich bei diesem Parameter um eine von vier Antwortmöglichkeiten: *weniger als 1 Woche* / *weniger als 1 Monat* / *1 bis 3 Monate* / *mehr als 3 Monaten*. Für die Abbildung auf der x-Achse werden diese Antwortmöglichkeiten auf durch Werte 1, 2, 3, und 4 kodiert.
- Eine Selbsteinschätzung im Umgang mit dem Werkzeug als eine von fünf Antwortmöglichkeiten: *sehr ungeübt* / *ungeübt* / *etwas erfahren* / *erfahren* / *sehr erfahren*. Diese werden für die Abbildung auf der x-Achse als Zahlenwerte 1 bis 5 kodiert.
- Die Erfahrung des Probanden im Einsatzfeld eines professioneller Projekte. Erfahrung bezeichnet hier Kenntnis des verwendeten Technologie-Stack und technologischen Umfeldes. Dieser Parameter wurde ursprünglich als Freitext erfasst und für die Abbildung in eine Skala von 0 bis 3 übersetzt. Dabei steht 1 für wenig, 2 für mittlere, 3 für umfassende Erfahrung in dem jeweiligen Projektkontext. Wurde das Werkzeug durch den Probanden nicht im Kontext eines professionellen Projektes eingesetzt, wird der Wert 0 verwendet.
- Die Erfahrung des Probanden im Einsatzfeld nicht-professioneller Projekte. Dieser Parameter ist analog zu dem Vorherigen: Es geht ebenfalls um technische Erfahrung, er wurde ebenfalls ursprünglich als Freitext erfasst und wird auf dieselbe Art in eine Skala von 0 bis 3 übersetzt.
- Die Information, ob sich der Proband aktiv oder gezielt in den Umgang mit dem jeweiligen Werkzeug eingearbeitet hat. Im Falle einer gezielten Einarbeitung wird der Wert 1 verwendet, ansonsten 0.

Auf der y-Achse werden die Antworten auf die Fragen 6 bis 8 abgebildet. Jede dieser Fragen besteht aus der Auswahl einer von fünf Antwortmöglichkeiten, sowie einem Freitextanteil. Für diese Abbildung werden jeweils die

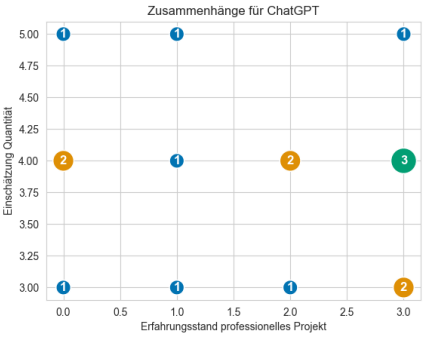
fünf Antwortmöglichkeiten in eine Skala von 1 bis 5 übersetzt, wie in Unterabschnitt 4.2.1 dargestellt. Dabei stellt jeweils 1 die negativste, 3 die neutralste und 5 die positivste Antwortmöglichkeit dar. Die Punkte stellen die Antworten einzelner Probanden dar. Die auf den Punkten abgebildeten Ziffern zählen Antworten zusammen, die auf dieselben Koordinaten fallen. Der Freitextanteil wird nicht berücksichtigt. Ausgehend von den sechs Kontext-Parametern, die jeweils auf die Antworten drei einordnenden Fragen abgebildet werden, ergeben sich 18 zu untersuchende Streudiagramme je Werkzeug. Abbildung 4.6 zeigt eine Auswahl von vier dieser Diagramme: Abbildung 4.6a zeigt die Abbildung des Parameters *Jahre als Software Engineer* auf die Antworten auf Frage 8 *Fähigkeiten/Lerneffekt* für ChatGPT. Abbildung 4.6b zeigt die Abbildung der Selbsteinschätzung der Probanden im Umgang mit GitHub Copilot auf die Antworten zu Frage 6 *Qualität*. Abbildung 4.6c stellt die Abbildung der Selbsteinschätzung an Erfahrung im Kontext professioneller Projekte auf die Antworten zu Frage 7 *Effizienz* für ChatGPT dar, während Abbildung 4.6d selbiges für GitHub Copilot zeigt. Diese vier Diagramme werden im Folgenden näher diskutiert. Eine Auswahl der verbleibenden 28 Diagramme hängt in Abschnitt A.3 an.



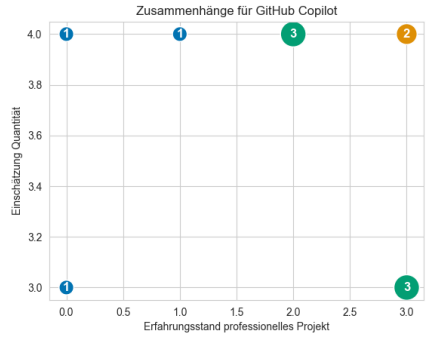
(a) Zusammenhänge für ChatGPT



(b) Zusammenhänge für GitHub Copilot



(c) Zusammenhänge für ChatGPT



(d) Zusammenhänge für GitHub Copilot

Abbildung 4.6: Abbildungen der Kontext-Parameter auf einige der einordnenden Fragen zu Qualität, Effizienz sowie Lerneffekt für ChatGPT und GitHub Copilot. Die Ziffern auf den Punkten fassen Daten zusammen, die auf dieselben Koordinaten fallen. Unterabschnitt 4.4.1 geht näher auf die Abbildungen ein.

Insofern in den erhobenen Daten Abhängigkeiten zwischen dem Kontext und den Antworten auf die einordnenden Fragen 6 bis 8 erkennbar sind, so würden diese als Muster in den Diagrammen hervortreten. Würde sich etwa die Beispiel-Hypothese aus der Einleitung dieses Unterabschnitts bestätigen, so wäre dies in Abbildung 4.6a zu bemerken. In diesem Fall wäre der durch die Probanden erlebte Fähigkeitsgewinn geringer, je mehr Jahre sie bereits als Software Engineers arbeiten. Das hätte zur Folge, dass Quadranten mit hohen X- sowie Y-Werten (Erfahrene Probanden, welche einen hohen Fähigkeitsgewinn wahrnahmen), als auch niedrigen X- und Y-Werten (Unerfahrene Probanden, welche einen niedrigen Fähigkeitsgewinn wahrnahmen), weniger stark mit Punkten besetzt wären. Stattdessen wären vor allem Punkte bei niedrigen X-Werten hohe Y-Werte zu finden, die von dort aus entlang positiver X-Richtung abfallen. Dieses Muster zeichnet sich nicht ab.

Aus Abbildung 4.6b lässt sich ableiten, inwiefern die Selbsteinschätzung im Umgang mit Copilot einen Einfluss auf die wahrgenommene Qualität der eigenen Arbeit mit Unterstützung des Werkzeugs hat. So ist etwa denkbar, dass Probanden, welche sich selbst als sehr erfahren im Umgang mit Copilot einschätzen, auch eine höhere Qualität in ihrer Arbeit mit dem Werkzeug erzielen. Diese Hypothese würde sich im Diagramm durch eine große Menge an Punkten mit numerisch ähnlichen X und Y Koordinaten abzeichnen (in denen die nummerische Differenz zwischen X und Y gering ist). In diesem Fall würde eine Steigung von X (der Selbsteinschätzung) mit einer Steigerung von Y (der wahrgenommenen Qualität) einhergehen. Gleichzeitig würden sich kaum Punkt finden, bei welchen sich die Koordinaten stark unterscheiden. Auch dieses Muster zeichnet sich nicht ab. Stattdessen sind die Punkte sehr gleichmäßig über das Diagramm verteilt.

Abbildung 4.6c und Abbildung 4.6d setzen die Selbsteinschätzung der Erfahrung im Kontext professioneller Projekte in Bezug zu den Antworten auf die Frage 7 *Effizienz* für ChatGPT und GitHub Copilot in Bezug. Für beide Werkzeuge sind die Punkte sehr gleichmäßig über die Diagramme verteilt. Es sind keine Gruppierung oder Verdichtung der Erfahrungen zu erkennen, die auf Abhängigkeiten hindeuten würden. Auch im Vergleich zwischen der Erfahrungen bei $X = 0$ (nicht im Kontext professioneller Projekte eingesetzt) und $X = 1$ *bis* 3 (Erfahrung im Kontext professioneller Projekte) sind keine Unterschiede zu erkennen.

Allgemein lassen die Ergebnisse nicht auf eine Abhängigkeit zwischen den erhobenen Kontext-Parametern und den Antworten auf die einordnenden Fragen 6 bis 8 schließen. Keines der Streudiagramme zeigt erkennbare und aussagekräftige Muster oder Gruppierungen. Das spricht dafür, dass die Effekte der Unterstützung durch KI-Werkzeuge im Software Engineering nicht pauschal von einzelnen Parametern abhängen, sondern sich nach der komplexen, individuellen Situation unterscheiden. Inwiefern sich bei einer größeren Datenmenge Muster abzeichnen würden — darüber kann keine Aussage getroffen werden. Im Kontext dieser Arbeit ist zu berücksichtigen, dass die individuelle Situation ausschlaggebend für den zu erwartenden Mehrwert der Unterstützung ist.

4.4.2 *Unterschiede nach Version von GPT*

Wie in Abschnitt 4.1 beschrieben, wurde ChatGPT von den meisten Probanden in der kostenfreien Variante verwendet — trotz Kostenübernahme von Accso. Beide Varianten ermöglichen die Verwendung des Modells GPT-3.5, während die kostenpflichtige Variante auch die Verwendung des Modells GPT-4 ermöglicht. Nach Angaben von OpenAI — der Firma hinter dem Chatbot — ist GPT-4 ein grundsätzlich kompetenteres, wenn auch langsameres Modell als GPT-3.5 [Ope23a]. Dieser Unterabschnitt untersucht, ob sich dieser Unterschied in den Erfahrungen der Probanden widerspiegelt. Dazu werden die Antworten der Probanden auf die einordnenden Fragen 6 bis 8 in Abhängigkeit von des verwendeten Modells gegenübergestellt. Tabelle 4.2 zeigt die Antworten in Abhängigkeit von der verwendeten Variante.

Verwendete Modelle	Nur GPT-3.5	GPT-3.5 und -4	Nur GPT-4
Anzahl der Erfahrungen	9 Erfahrungen	3 Erfahrungen	4 Erfahrungen
6. Qualität	ø 4	ø 4	ø 3,5
	$\sigma \approx 0,53$	σ 0	$\sigma \approx 0,58$
7. Effizienz	ø $\approx 3,67$	ø 4	ø 4,25
	$\sigma \approx 0,71$	σ 1	σ 0,5
8. Fähigkeiten / Lerneffekt	ø $\approx 2,89$	ø 3	ø 4,25
	$\sigma \approx 1,45$	σ 1	σ 0,5

Tabelle 4.2: Gegenüberstellung der durschnittlichen Bewertung ø und der Standardabweichung σ der Antworten auf die einordnenden Fragen 6 bis 8 in Abhänigkeit der verwendeten Modelle.

Während die Antworten auf Fragen 6 und 7 keine eindeutigen Unterschiede zeigen, scheint die Verwendung von GPT-4 einen positiven Einfluss auf die Antworten auf Frage 8 zu haben. Im Vergleich zwischen GPT-3.5 über beide Modelle hin zu GPT-4 steigt die durchschnittliche Bewertung von knapp unter 3 auf 4,25. Zugleich fällt die Standardabweichung von 1,45 auf 0,5. Diese Zahlen deuten auf einen Fähigkeitsgewinn durch die Unterstützung von GPT-4 hin. Von GPT-4 unterstütze Probanden sahen sich (im Vergleich zu GPT-3.5) eher in der Lage, Tätigkeiten auszuüben, welche sie ohne das Werkzeug nicht hätten ausführen können. Diese Schlussfolgerung deckt sich mit der Aussage von OpenAI, dass GPT-4 ein kompetenteres Modell ist. Allerdings ist die Aussagekraft dieser Schlussfolgerung durch die geringe Anzahl der GPT-4-benutzenden Probanden stark begrenzt. Darüber hinaus ist die Kompetenz der Modelle teilweise sehr inkonsistent, wie in Unterabschnitt 2.4.3 diskutiert. Es ist nicht ausgeschlossen, dass die Probanden, welche ihre Erfahrungen mit GPT-4 zufällig zu einem Zeitpunkt sammelten, zu welchem das Modell besonders kompetent war. Insgesamt ist es auf Basis dieser Untersuchungen nicht möglich, klare Aussagen über die Unterschiede zwischen den beiden Modellen im Kontext des Software Engineerings zu treffen.

4.4.3 *Potenziale für die Zukunft*

Der Fokus dieser Arbeit und der in Kapitel 3 vorgestellten Potenzialanalyse liegt in der Gegenwart. Es geht um den Mehrwert, den KI-basierte Werkzeuge heute bereits für das Software Engineering bieten. Über diese Zielsetzung hinaus bietet dieser Abschnitt einen Exkurs in die Zukunft. Es werden die Potenziale diskutiert, welche die Probanden auf Basis ihrer Erfahrung im Umgang mit den Werkzeugen für deren Einsatz in der Zukunft. Auch wenn die Antworten auf Frage 9 des Fragebogens als Grundlage hierfür dienen, handelt es sich bei dem Inhalt dieses Unterabschnittes um Spekulation.

Zusammengefasst sehen neun Stimmen kleinere Verbesserungen in Reichweite von ChatGPT. Es ist davon auszugehen, dass allgemeine Verbesserungen und Weiterentwicklungen des Chatbots im Bereich des Möglichen sind. Für eine Verbesserung des Workflows ist eine Integration in die Entwicklungsumgebung ein unausweichlicher Schritt. Viele der Vorteile von ChatGPT werden durch die Notwendigkeit zurückgehalten, dem Chatbot in umfangreichen Prompts den Kontext der Problemstellung zu erklären. Wäre der Chatbot in die IDE integriert und könnte den Kontext selbst ableiten, würde dieser Nachteil wegfallen. Das bereits angekündigte, jedoch noch nicht frei verfügbare, Werkzeuge GitHub Copilot X[1] verspricht genau diese Integration. Eine Untersuchung dieses Werkzeugs könnte Aufschluss darüber geben, inwiefern dieses Potenzial sich bestätigt. Ein Hindernis bei der Integration in die IDE stellt der Geheimnisschutz dar. Dabei handelt es sich jedoch nicht um ein großes Hindernis, da GitHub Copilot — nicht X, sondern die aktuelle, im Rahmen dieser Arbeit untersuchte Version — bereits heute diese Einschränkung aufweist und trotzdem bereits eingesetzt wird.

Größere Potenziale sehen zusammengefasst 11 Stimmen in umfassender Unterstützung des Chatbots. Inwiefern die Einarbeitung neuer Mitarbeiter in Zukunft zunehmend mithilfe des Chatbots erfolgen kann, ist schwer abzuschätzen. Doch die Untersuchungsergebnisse deuten darauf hin, dass der Chatbot bereits heute beim Lernen und der Einrichtung von Entwicklungsumgebungen helfen kann. Beide Punkte sind elementar für die Einarbeitung neuer Mitarbeiter. Ob das Testen von Code in Zukunft umfassender durch den Chatbot unterstützt werden kann, hängt wahrscheinlich unter anderem mit der größe des Kontextes zusammen, welchen der Chatbot verarbeiten kann. Insofern der Chatbot den zu testenden Code, die einzuhaltenden Anforderungen und weiter Parameter bei der Generierung von automatisieren Tests im Kontext halten und verarbeiten kann, könnte dessen Unterstützung zu einem massiven Zeitgewinn beitragen. Weitergehende Unterstützung in der Softwarearchitektur ist ebenfalls ein großes Potenzial, welches sich bereits heute andeutet. Im Rahmen der Untersuchungen haben die Probanden gute Erfahrungen damit gemacht, ChatGPT als Wissensressource zu verwenden, welche Hinweise zu individuellen Situationen geben kann und damit den herkömmlichen Quellen etwas voraus hat. Inwieweit eine Weiterentwicklung des Chatbots seine Kompetenz in diesem Gebiet steigern kann, ist schwer abzuschätzen, doch scheint

1 https://github.com/features/preview/copilot-x (Zuletzt aufgerufen am 29.09.2023)

hier echtes Potenzial zu bestehen. Ob ein Paradigmenwechsel im Software Engineering durch das Aufkommen LLM-basierter Chatbots eingeleitet wird, ist das aktuell am schwersten einzuschätzende Potenzial, welches von den Probanden genannt wurde. In einem solchen Paradigmenwechsel könnte die Rolle von Engineers sich weg von dem Schreiben von Code und hin zum Gegenprüfen dessen und dem Fokus auf Requirements-Engineering/-Management verschieben. Zwar scheinen große Sprachmodelle ein erster Schritt in die Richtung der Automatisierung von Kopfarbeit zu sein, doch jede Spekulation über dieses Potenzial ist mit großer Unsicherheit behaftet.

Insgesamt sechs Stimmen sehen die Potenziale der Werkzeuge durch einige Limitationen zurückgehalten. Neben vielversprechenden Potenzialen für die Zukunft ist es ebenfalls denkbar, dass der Einsatz von LLM-basierten Chatbots auf wenige Anwendungsfälle beschränkt bleibt. Herausforderung könnten dabei ein Mangel an Kreativität und an tatsächlichem Verständnis sein, welchen KI womöglich nie aufweisen wird (siehe Unterabschnitt 2.3.4). Inwiefern diese Limitationen tatsächlich bestehen, kann in dieser Arbeit nicht beantwortet werden.

Im Fall von GitHub Copilot sehen sechs Stimmen Potenzial für generell sinnvollere Vorschläge, welche die Unterstützung des Werkzeugs noch effizienter machen. Auch hier scheinen kleinere Weiterentwicklungen dieser Art in Reichweite, ohne dass große Änderungen an der Wesensart des Werkzeugs notwendig wären. Über die Untersuchungen waren die Probanden sehr zufrieden mit der Art der integrierten Unterstützung, welche GitHub Copilot bietet. Diese Zufriedenheit geht so weit, dass für die Arbeit mit dem Werkzeug kaum konkrete Lessons Learned angegeben wurden. Auch wenn in den Freitextantworten häufig bemängelt wurde, dass die Werkzeuge an Komplexität teilweise noch scheitern, scheint hier Potenzial für die Zukunft zu bestehen.

Zwei Stimmen halten dieses Verbesserungspotenzial durch fehlendes Domänenwissen und fachlichem Verständnis limitiert. Wie oben erwähnt ist unklar, inwiefern es KI überhaupt möglich ist, echtes Verständnis zu entwickeln. Trotzdem ist nicht von der Hand zu weisen, dass Copilot auch in seiner aktuellen Form bereits heute und ohne tatsächliches fachliches Verständnis einen Mehrwert für das Software Engineering darstellt.

Alles in allem sind die Potenziale beider Werkzeuge für die Zukunft zwar schwer einzuschätzen, doch scheint es wahrscheinlich, dass sich der Mehrwert, welche die Werkzeuge bereits heute bieten, in Zukunft noch steigern wird.

4.4.4 *Rückfragen*

Im Rahmen der Auswertung des Fragebogens kamen zwei Punkte auf, welche potenzielle Lessons Learned zu sein schienen, die jedoch nicht auf Basis der gesammelten Daten eindeutig bestätigt werden konnten. Um diese Punkte näher zu untersuchen wurde im Nachgang an die Interviews ein Online-Fragebogen an die Probanden der Pilotuntersuchungen versendet. Die Probanden, mit welchen das Interview auf Englisch geführt wurde, erhielten die

Rückfragen ebenfalls in englischer Sprache. Insgesamt nahmen 15 der 19 Probanden an der Rückfragerunde teil. Die entsprechenden Übersetzungen hängen in Abschnitt A.4 an. In diesem Unterabschnitt ist für jeden Punkt jeweils der Kontext und die Rückfrage selbst vorgestellt, sowie die Antworten der Probanden und was sich daraus ableiten lässt.

Timebox für die Arbeit mit ChatGPT

Diese Rückfrage basiert auf einer Einzelmeinung aus den Freitextantworten auf Frage 12 *Lessons Learned*. Wie in Unterabschnitt 4.2.3 beschrieben, werden Einzelmeinungen bei der Auswertung der Freitextantworten standardmäßig aufgrund des hohen Umfangs der Antworten und deren beschränkter Aussagekraft eigentlich nicht berücksichtigt. Diese Einzelmeinung ist jedoch eine direkte Gegenmaßnahme gegen ein mehrfach erwähntes Problem im Umgang mit dem ChatGPT: Bei der iterativen Arbeitsweise mit dem Chatbot, bei welcher Prompts in mehreren Iterationen erarbeitet werden und viel mit dem Chatbot kommuniziert wird, kann es vorkommen, dass sich die Unterhaltung mit dem Chatbot im Kreis dreht, ohne bei einem Ergebnis näherzukommen. Die Einzelmeinung empfiehlt, sich bei der Arbeit mit ChatGPT eine Timebox zu setzen, um in solchen Fällen zu verhindern, dass viel Zeit verschwendet wird. Sobald die Timebox erreicht ist, sollte auf andere Quellen, Werkzeuge oder Hilfen zurückgegriffen werden. Auch wenn dieses Vorgehen logisch erscheint, ist es als Einzelmeinung nicht fundiert genug, um als Best Practice in das Nutzungskonzept aufgenommen zu werden. Deshalb wird diese Empfehlung in der Rückfragerunde überprüft. Dazu wurde den Probanden, welche ChatGPT verwendeten, die folgende Frage gestellt:

> Im Umgang mit ChatGPT kann das Problem auftreten, dass sich die Unterhaltung mit dem Chatbot im Kreis dreht, ohne dabei einem Ergebnis näherzukommen. **Basierend auf deiner Erfahrung** im Umgang mit **ChatGPT**, würdest du empfehlen, sich bei der Arbeit mit dem Chatbot eine Timebox zu setzen, um in solchen Fällen zu verhindern, dass viel Zeit verschwendet wird?

Die Antwortmöglichkeiten waren *Ja*, *Nein* und *Keine Aussage*. Von den 12 Probanden, welche ChatGPT erprobten, antworteten 9 mit *Ja*, 2 mit *Nein* und 1 mit *Keine Aussage*. Diese Rückmeldung ist eindeutig und deutet darauf hin, dass das Setzen einer Timebox eine valide Gegenmaßnahme gegen das Risiko darstellt, bei im Kreis drehenden Unterhaltung mit dem Chatbot Zeit zu verlieren. Diese Best Practice wird deshalb in das Nutzungskonzept aufgenommen.

„Comment Driven“ mit GitHub Copilot arbeiten

Genau wie die vorherige Rückfrage basiert auch diese auf einer Einzelmeinung aus den Freitextantworten auf Frage 12 *Lessons Learned*. Auch sie wird betrachtet, da sie eine direkte Gegenmaßnahme gegen ein mehrfach erwähntes Problem im Umgang mit dem Werkzeug ist. Copilot kann nur dann helfen,

wenn ein erster Schritt bereits getan ist — etwa erste Zeilen Code geschrieben sind. Darüber hinaus scheitert der Assistent an komplexeren Codeabschnitten eher und wird durch fehlendes fachliches Verständnis zurückgehalten. Die Einzelmeinung empfiehlt, „Comment Driven" zu arbeiten. Dabei wird für die Implementierung eines Codeabschnitts zunächst ein Kommentar in Textform geschrieben, welcher die Funktionalität des zu implementierenden Codeabschnitts beschreibt. Anschließend wird der Cursor in die nächste Zeile bewegt und der Codeabschnitt mit Unterstützung von Copilot implementiert. Auf diese Weise hat Copilot mehr Kontext über die zu implementierende Funktionalität und scheint bessere Vorschläge zu unterbreiten. Der Ansatz funktioniert dabei ähnlich wie die Verwendung von Copilot als Alternative zur Websuche, welche in den Freitextantworten zu Frage 8 beschrieben wurde. Dieses Vorgehen erscheint sinnvoll, ist jedoch als Einzelmeinung nicht fundiert genug zur Aufnahme als Best Practice in das Nutzungskonzept. Deshalb wurde auch diese Empfehlung in der Rückfragerunde anhand folgender Frage überprüft:

> GitHub Copilot scheint bei komplexeren Codeabschnitten weniger gute Vorschläge zu unterbreiten. Darüber hinaus kann Copilot erst dann helfen, wenn bereits ein erster Schritt in der Implementierung getan ist (etwa erste Zeilen Code geschrieben sind). Diesen Herausforderungen kann womöglich begegnet werden, indem „Comment Driven" gearbeitet wird. Dabei wird für die Implementierung eines Codeabschnitts zunächst ein Kommentar in Textform geschrieben, welcher die zu implementierende Funktionalität beschreibt. Anschließend wird der Cursor in die nächste Zeile bewegt und der Codeabschnitt mithilfe von Copilot implementiert. Auf diese Weise steht Copilot mehr Kontext zur Verfügung.
> **Basierend auf deiner Erfahrung** im Umgang mit **Copilot**, würdest du empfehlen, „Comment Driven" zu arbeiten?

Die Antwortmöglichkeiten waren *Ja*, *Nein* und *Keine Aussage*. Von den 9 Probanden, welche GitHub Copilot erprobten, antworteten 6 mit *Ja*, 2 mit *Nein* und 1 mit *Keine Aussage*. Wenn diese Rückmeldung auch nicht eindeutig ist, so deutet sie tendenziell doch auf Zustimmung hin. Da die Rückfragen keinen Freitextanteil beinhalten, ist eine Interpretation hier schwierig. Eine Vermutung für den Ursprung der Gegenmeinungen könnte jedoch ein Geschwindigkeitsverlust sein, der mit der „Comment Driven" Arbeitsweise einhergeht, da der jewelige Codeabschnitt zunächst in Textform beschrieben werden muss. Die Empfehlung, „Comment Driven" mit Copilot zu arbeiten wird als Best Practice in das Nutzungskonzept aufgenommen, jedoch mit dem Hinweis, dass diese Empfehlung nicht für alle Probanden einen Mehrwert darstellt.

4.4.5 *Kritik und Limitationen*

Dieser Unterabschnitt diskutiert Limitationen, welche die Validität und Aussagekraft der Ergebnisse einschränken können. Die Limitationen entspringen

vielen Quellen und sind relevant für den Kontext, in welchem die Ergebnisse dieser Arbeit stehen. Dabei stehen die Einschränkungen der Validität der Ergebnisse nicht entgegen, sondern ordnen diese lediglich ein.

Eine zentrale Einschränkung findet sich im Umfang der Untersuchungen. Die hier präsentierten Ergebnisse basieren lediglich auf einer ersten Untersuchung in dem aktiven Forschungsfeld des Einsatzes LLM-basierter Werkzeuge im Software Engineering. Insgesamt nahmen nur 19 Probanden an den Untersuchungen teil. Pro Werkzeug waren es sogar noch weniger Probanden, was die statistische Aussagekraft der Ergebnisse einschränkt. Ebenso wurden nur zwei der 16 evaluierten Werkzeuge erprobt und viele weitere Werkzeuge nicht in die Evaluation aufgenommen. Darüber hinaus lag der Fokus auf einem ersten Vorstoß in das Forschungsfeld, um einen Überblick über die Möglichkeiten und Grenzen der Werkzeuge im gesamten Prozess zu erhalten. Es wurde den Probanden keine Vorgaben bezüglich der Anwendungsfälle der Werkzeuge gemacht, was dazu geführt haben könnte, dass die einzelnen Anwendungsfälle des SE nicht eingehend untersucht wurden.

Eine weitere Einschränkung findet sich im Design der Untersuchungen. Die Evaluation von Qualität und Effizient im Software Engineering selbst ist eine grundsätzliche Herausforderung. In diesem Kontext ist es schwierig, die Genauigkeit und Aussagekraft der Selbsteinschätzungen der Probanden zu beurteilen. Auch wurden für Benotung der Erfahrungen der Probanden mit den Anwendungsfällen auch Noten zwischen den vorgesehenen Werten zugelassen, etwa 3, 5. Das geschah, um die Antworten der Probanden möglichst präzise abbilden zu können, doch sind diese Zwischenantworten auch ein Indiz dafür, dass die Granularität der Anwendungsfälle unter Umständen nicht ausreichend genug war.

Auch die Methodik der Auswertung wirft Limitationen auf. So wurden etwa aufgrund der geringen Datenmenge nur der Durchschnitt und die Standardabweichung als statistische Kennzahlen verwendet. In diesem Kontext hat dieses Vorgehen seine Berechtigung, doch sind dadurch keine statistisch signifikanten Aussagen möglich. Das Zusammenfassen der Freitextantworten (siehe Unterabschnitt 4.2.3) geschah händisch und ohne die Verwendung literaturwissenschaftlicher Methoden, was ebenfalls der kleinen Datenmenge geschuldet ist. Umfassende Methoden wären auch dort zu weit gegangen, doch ist die händische Auswertung eine mögliche Fehlerquelle in der Auswertung. Auch die Untersuchung auf statistische Zusammenhänge zwischen den, durch den Fragebogen erfassen, Variablen (siehe Unterabschnitt 4.4.1) geschah ohne den Einsatz statistischer Methoden und ist somit eine potenzielle Fehlerquelle.

Darüber hinaus erhebt diese Arbeit keinen Anspruch auf Vollständigkeit. Die identifizierten Anwendungsfälle und Best Practices sind aller Wahrscheinlichkeit nach unvollständig. Es ist nicht ausgeschlossen, dass weiter Untersuchungen Einzelmeinungen bestätigen, welche hier aufgrund mangelnder Indizien nicht als Best Practice aufgenommen wurden. Zwar haben jegliche der präsentierten Resultate Rückhalt in der Evidenz, doch handelt es sich nicht um eindeutig bewiesene Fakten. So könnten Vorgehensweise, die hier als Best

Practices identifiziert wurden, in weiteren Untersuchungen mit mehr Daten als weniger hilfreich erkannt werden. Auch könnten Zusammenhänge und Abhängigkeiten zwischen den im Fragebogen erfassten Variablen bestehen, welche erst auf Basis von mehr Daten und durch die Verwendung statistischen Methoden zutage treten.

Nicht nur das maschinelle Lernen beinhaltet Bias. Auch in den hier vorgestellten Untersuchungen kann Bias nicht ausgeschlossen werden. Das beginnt etwa mit der Auswahl der Probanden, wobei deren Interesse an der Teilnahme eine Voraussetzung war. Es ist denkbar, dass die Antworten von interessierten Probanden im Vergleich zu nicht-interessierten Probanden positiver ausfallen. Genauso könnten Probanden während der Erprobung der Werkzeuge die wenig-erfolgreichen Anwendungsfälle weniger ausführlich erprobt haben, sodass diese sich in geringerem Maße in den Antworten widerspiegeln. Auch beinhalten die, durch den Fragebogen erhobenen, Selbsteinschätzungen der Probanden inhärent Bias. Ebenso beinhalten die Interviews, welche durch den Autor dieser Arbeit geführt wurden, Bias, da sie nicht immer identisch abgelaufen sind und so keine hundertprozentige Neutralität gewährleisten können.

Eine weitere Quelle für Abweichungen in den Untersuchungsergebnissen sind die in Unterabschnitt 2.4.3 erwähnten Schwankungen in den Fähigkeiten LLM-basierter Werkzeuge. Diese Schwankungen könnten über mehrere Wochen und Monate hinweg dazu führen, dass eine Probandin, die ein bestimmtes Werkzeug in einem bestimmten Anwendungsfall erprobt, zu unterschiedlichen Zeiten unterschiedlich erfolgreich damit ist. Darin liegt eine weitere potenzielle Fehlerquelle und auch eine Erklärung für die Unterschiede in den Erfahrungen der Probanden.

4.4.6 *Fazit*

Insgesamt zeigen die Untersuchungsergebnisse, dass viele Anwendungsfälle im Software Engineering bereits heute vom Einsatz LLM-basierter Werkzeuge profitieren. Beide Werkzeuge unterstützen die Geschwindigkeit der Arbeit, wobei ChatGPT unter Umständen sogar dabei helfen kann, eine bessere Lösung zu erzielen. Dabei sind die Werkzeuge jedoch nicht in allen Anwendungsfällen und Situationen in gleichem Maße hilfreich: Der Mehrwert ist häufig von der individuellen Situation abhängig, etwa von Unterschieden im Projektkontext, von den verwendeten Technologien, der Vorerfahrung der Engineers oder persönlicher Präferenz. Es wurden jedoch keine Zusammenhänge zwischen den erfassten Variablen der individuellen Situation und dem von den Probanden erlebten Mehrwert gefunden. Auch scheint die bei ChatGPT verwendete Version des Modells keinen großen Unterschied zu machen. Alles in allem sind diese Ergebnisse jedoch nur ein erster Schritt in das Forschungsfeld, welcher im Umfang und in der Aussagekraft eingeschränkt ist. Die in Abschnitt 3.4 aufgestellte Vermutung, die Unterstützung der Werkzeuge würde vorrangig in den Domänen 3 bis 5 zutage treten, hat sich im Großen und Ganzen bestätigt. Bei der Auswertung der Ergebnisse waren darüber hinaus die

Freitextantworten der Probanden besonders hilfreich, um die quantitativen Ergebnisse inhaltlich einordnen zu können.

5

ERGEBNISSE — NUTZUNGSKONZEPT UND BEST PRACTICES

In diesem Kapitel werden die konkreten Ergebnisse der Untersuchungen vorgestellt: ein Nutzungskonzept, inklusive Best Practices, für die Nutzung zweier Kategorien von KI-Werkzeugen im Software Engineering — „Intelligente“ Chatbots und KI-Pair-Programmer. Zunächst werden in Abschnitt 5.1 ein allgemeines Nutzungskonzept, sowie die betrachteten Werkzeuge vorgestellt. Abschnitt 5.2 führt Einschränkungen auf, die bei der Verwendung der Werkzeuge zu beachten sind. Anschließend präsentieren Abschnitt 5.3 und Abschnitt 5.4 detailliertere Nutzungskonzepte für ChatGPT und GitHub Copilot. Abschnitt 5.5 gibt eine Übersicht über die Anwendungsfälle, in welchen die Werkzeuge bereits heute effektiv unterstützen können. Zuletzt sind in Abschnitt 5.6 und Abschnitt 5.7 Best Practices für die Nutzung von ChatGPT und Copilot aufgeführt.

5.1 NUTZUNGSKONZEPT ALLGEMEIN

Der Arbeitsprozess der Software Engineers bleibt auch mit Unterstützung durch KI-Werkzeuge weitestgehend unverändert. Die einzige Änderung ist, dass ihnen Werkzeuge aus den beiden Kategorien „intelligente“ Chatbots und KI-Pair-Programmer zur Verfügung stehen. Diese Werkzeuge bergen das Potenzial, durch ihre Unterstützung die Qualität und Effizienz der Arbeit zu steigern. Dabei ist nicht nur die Qualität des Resultats positiv betroffen, sondern auch der Komfort der Engineers — wenn auch die Ausgaben der Werkzeuge genau überprüft werden müssen, um die Codequalität zu erhalten. Ein Geschwindikeitszuwachs zeichnet sich vor allem bei repetitiven Aufgaben und der Verfügbarkeit von Wissen ab. Die Werkzeuge können auch zu neuen Lösungen oder Ansätzen inspirieren, welche die Engineers andernfalls nicht in Betracht gezogen hätten.

„Intelligente“ Chatbots sind passive Werkzeuge. Sie stehen auf Abruf zur Verfügung — etwa in einem weiteren Browser-Tab oder Fenster. Wann immer eine Engineer es wünscht, kann sie mit dem Chatbot interagieren, indem sie eine Nachricht — ein Prompt — an das Werkzeug schickt und dessen Antwort erhält. Im Kontrast dazu laufen KI-Pair-Programmer aktiv in der Entwicklungsumgebung mit und unterstützen dauerhaft. Während dem Schreiben von Code, Dokumentation oder Text schlagen sie laufend Vervollständigungen vor, welche die Engineer annehmen oder ignorieren kann. So hat die Engineer minütlich viele kleine Interaktionen mit dem Werkzeug, ohne dabei den eigenen Workflow verlassen zu müssen. Da ChatGPT und GitHub Copilot als exemplarische Vertreter dieser Kategorie untersucht wurden, wird im Folgenden konkret von diesen Werkzeugen gesprochen.

Um von den oben genannten Vorteilen zu profitieren, müssen die Engineers nicht zwangsläufig Werkzeuge aus beiden Kategorien verwenden. Wie in Abschnitt 4.4 gezeigt, variiert die individuelle Erfahrung mit den Werkzeugen stark. Unterschiede im Projektkontext, die verwendeten Technologien, die eigene Erfahrung und persönliche Präferenz schaffen ein individuelles Arbeitsumfeld, in welchem die Werkzeuge nicht immer im selben Maße unterstützen. Mit jedem verwendeten Werkzeug summieren zusätzlich sich die Vorteile und Nachteile dieser Unterstützung auf. Die Abwägung, welche Werkzeuge wie verwendet werden hat großen Einfluss auf die eigene Arbeit und kann nur von den Engineers und selbst getroffen werden. Um diese Entscheidung sinnvoll treffen zu können und das volle Potenzial der Unterstützung abrufen zu können, braucht ein Engineer einen guten Überblick über die Kompetenzen der Werkzeuge und über die Anwendungsfälle, in denen die Werkzeuge hilfreich sein können. Einen solchen Überblick bietet dieses Nutzungskonzept.

5.2 EINSCHRÄNKUNGEN UND HINWEISE

Während der Einsatz von KI-Werkzeugen im Software Engineering viele Vorteile bietet, so steht die Nutzung der Werkzeuge auch im Kontext der folgenden Einschränkungen. Diese gelten für den Umgang mit beiden Kategorien von Werkzeugen.

Kritische Prüfung der Ausgaben

Sowohl ChatGPT als auch GitHub Copilot machen Fehler. Als Werkzeuge, die auf großen Sprachmodellen und damit auf maschinellem Lernen basieren, sind Fehler unvermeidbar (siehe Unterabschnitt 2.1.3). Das macht es zur Notwendigkeit, jede Ausgabe der Werkzeuge kritisch zu betrachten — gerade, da Fehler unter Umständen nicht offensichtlich sind, sondern auf den ersten Blick korrekt erscheinen. Mangelnde Prüfung kann unter anderem zu Sicherheitslücken, Performance-Problemen, schlechter Wartbarkeit und generell allen Risiken im SE führen. Problematisch ist hier die menschliche Neigung zum sogenannten *Confirmation Bias*, also der Neigung zur Bevorzugung von Informationen, welche die eigene Meinung bestätigen. In der Erwartung, die Ausgaben der Werkzeuge sein korrekt, entsteht so leicht die Neigung, Fehler zu übersehen. Jede Entwicklerin ist selbst für den Code verantwortlich, den sie schreibt, aber auch für den Code, den sie von einem KI-Werkzeug akzeptiert. Wer kein Experte auf einem Gebiet ist, kann Fehler unter Umständen gar nicht erst erkennen.

Die Fehleranfälligkeit der Werkzeuge ist allerdings kein Hindernis für deren Nutzung. Auch bei alternativen Informationsquellen sind Skepsis und kritische Prüfung notwendig. Etwa bei der Websuche, der Verwendung von Stackoverflow oder auch Ratschlägen von Kollegen. Keine Information sollte blind übernommen werden. Als eine weitere Informationsquelle, die gegenüber ihrer Alternativen einige gravierende Vorteile bietet, bieten die Werkzeuge trotz ihrer Fehleranfälligkeit einen Mehrwert für das SE. Alles in allem gilt

es, die Werkzeuge wachsam zu nutzen und jegliche Ausgabe zu überprüfen — genau wie Verwendung von ChatGPT oder GitHub Copilot.

Sicherstellung von Daten- und Geheimnisschutz

Das Einhalten des Daten- und Geheimnisschutzes war bei der Aufstellung der Probandengruppe in vielen Projekten ein Hindernis. Vor der Nutzung jeglicher an das Internet angeschlossenen Werkzeuge ist es zwingend erforderlich, den Daten- und Geheimnisschutz einzuhalten. Ob künstliche Intelligenz zum Einsatz kommt, spielt dabei keine Rolle. Wichtig ist das Bewusstsein, dass jegliche Daten von den verarbeitenden Anbietern potenziell eingesehen, gespeichert und weiterverwendet werden können. Im Falle von ChatGPT besteht sehr klare Kontrolle darüber, welche Daten an den Chatbot übermittelt werden. Im Falle von GitHub Copilot ist es andererseits deutlich intransparenter, welche Teile der Codebase im Detail für die Vorschläge verwendet werden. Hier ist sinnvoll, Copilot ausschließlich dann zu verwenden, wenn alle der IDE zugänglichen Daten keinem besonderen Schutz unterliegen. Die KI-Werkzeuge sollten ausschließlich mit Vorsicht und unter Absprache mit Accso, ihren Kunden und Betroffenen genutzt werden.

Bewusstsein über Bias und Trainingsdaten

Als Werkzeuge, die auf maschinellem Lernen basieren, sind ChatGPT und GitHub Copilot unausweichlich von Bias betroffen. Für die Engineers ist nicht ersichtlich, wie die Trainingsdaten zusammengesetzt sind oder auf welche Art die Modelle trainiert wurden. Im Umgang mit den Werkzeugen müssen die Engineers sich des Bias bewusst sein. Sie müssen davon ausgehen, dass die Werkzeuge zum Allgemeinen und zu Klischees neigen. Eine weitere Einschränkung ist die zeitliche Begrenzung der Trainingsdaten. Die Sprachmodelle, auf welchen sowohl ChatGPT als auch GitHub Copilot basieren, wurden mit Daten trainiert, die bis zu einem gewissen Zeitpunkt gesammelt wurden. Informationen, die nach diesem Zeitpunkt entstanden sind, sind den Werkzeugen somit vollkommen unbekannt. Für das Software Engineering bedeutet dies konkret: Den Werkzeugen fehlen Informationen über aktuelle technische Entwicklungen, etwa Neuerungen in Programmiersprachen, Frameworks oder APIs.

5.3 NUTZUNGSKONZEPT CHATGPT

ChatGPT ist ein „intelligenter“ Chatbot. Als solcher steht das Werkzeug auf Abruf zur Verfügung und kann via Prompts angesprochen werden, wann immer der Engineer es wünscht. Dafür verlässt der Engineer seinen Workflow und wechselt in den Browser, in welchem ChatGPT läuft. Dass der eigene Workflow verlassen werden muss, um mit dem Werkzeug zu interagieren, macht die Fähigkeit einzuschätzen, in welchen Szenarien das Werkzeug ge-

winnbringend eingesetzt werden kann, zu einer wichtigen Kompetenz im Umgang mit dem Chatbot. ChatGPT wird auf diese Weise ähnlich wie eine herkömmliche Suchmaschine verwendet, um etwa einer Fehlermeldung nachzugehen oder Syntax nachzuschlagen. Darüber hinaus bietet der Chatbot jedoch auch Funktionalität, welche über das Suchen nach Informationen hinaus geht. Er kann beispielsweise zum Erklären von individuellem Quellcode oder dem Generieren von benutzerdefinierten Testdaten aufgefordert werden.

Dabei sind folgende Effekte zu erwarten: Wie in Abschnitt 4.4 gezeigt, variiert die individuelle Erfahrung mit ChatGPT stark. Tendenziell scheint die Unterstützung des Chatbots jedoch sowohl die Qualität als auch die Effizienz der eigenen Arbeit zu erhöhen. Die Qualität scheint dabei besonders dann von der Unterstützung zu profitieren, wenn die Werkzeuge in Umgebungen und mit Programmiersprachen eingesetzt werden, mit welchen der Engineer nicht vertraut ist. Auch kann der Chatbot zu Ideen und Ansätzen inspirieren, welche der Engineer nicht selbst in Betracht gezogen hätte und so ein besseres Resultat ermöglichen. In jedem Fall bleibt die Qualität aber nur gewahrt, solange die Ausgaben des Chatbots gewissenhaft überprüft werden. Der Geschwindigkeitszuwachs zeichnet sich vor allem im Generieren von Code — speziell Boilerplate Code, sowie erste lauffähige Versionen von Features und Projekten — in der Analyse von Fehlermeldungen und in der Verfügbarkeit von Informationen ab, welche im Falle einer Websuche händisch gefiltert und sortiert werden müssten. Der Effizienzgewinn wird dadurch zurückgehalten, dass teilweise viel Kontextinformation an den Chatbot übermittelt werden muss, und dass die Ausgaben des Chatbots genau überprüft werden müssen. In einigen Fällen ermöglicht die Unterstützung von ChatGPT sogar einen Fähigkeitszuwachs. So kann eine Engineer mithilfe des Chatbots etwa im Umgang mit Technologien oder Programmiersprachen rasch arbeitsfähig werden, mit welchen sie zuvor nicht vertraut war. Auch das Lernen von neuen Programmiersprachen oder neuen Aspekten bekannter Programmiersprachen kann der Chatbot unterstützen. Den Fähigkeitszuwachs erfahren jedoch nicht alle Engineers. Wer davon profitiert scheint sehr individuell zu sein.

Insgesamt wird die Arbeit mit ChatGPT als sehr komfortabel und auch für Nicht-Experten zugänglich wahrgenommen. Dem Komfort abträglich ist allerdings die fehlende Verbindung zwischen Code-Basis und Chatbot. Als Konsequenz dessen ist es teilweise erforderlich, dem Chatbot viel Kontextinformation in den Prompts händisch zu übermitteln. Auch negativ ins Gewicht fällt die Unzuverlässigkeit und Fehleranfälligkeit von ChatGPT. Ein großes Thema sind dabei Halluzinationen, bei welchen der Chatbot etwa Funktionssignaturen oder Bibliotheken frei erfindet. Eine weitere Limitation sind die zeitlich begrenzten Trainingsdaten. So verfügt der Chatbot nicht über aktuelles Wissen, etwa zu Neuerungen in APIs oder Programmiersprachen. Wie in Unterabschnitt 4.4.2 diskutiert, lässt sich auf Basis der Untersuchungen keine Aussage über mögliche Unterschiede im Mehrwert zwischen dem kostenfrei zugänglichen Modell GPT 3.5 und dem kostenpflichtig-exklusiven Modell GPT 4 Variante von ChatGPT feststellen. Für dieses Nutzungskonzept bedeutet dies,

dass die Unterstützung von ChatGPT in jeglicher Form und unabhängig des verwendeten Modells wertvoll ist.

5.4 NUTZUNGSKONZEPT GITHUB COPILOT

GitHub Copilot ist ein KI-Pair-Programmer. Als solcher läuft das Werkzeug aktiv in der Entwicklungsumgebung mit und unterstützt dauerhaft, ohne dass der Engineer seinen Workflow verlassen muss. Diese Unterstützung geschieht durch kontinuierlich unterbreitete textuelle Vorschläge, welche den Inhalt des Editors an der Position des Cursors vervollständigen. Diese Vorschläge basieren auf der eigenen Code-Basis, wobei jedoch nicht ersichtlich ist, welche Teile des Codes genau verwendet werden. Der Engineer muss laufend entscheiden, ob er die Vorschläge annimmt oder ignoriert. Auf diese Weise kommt Copilot ähnlich wie die Autocomplete-Features moderner IDEs zum Einsatz. Ähnlich wie zwischen ChatGPT und herkömmlichen Suchmaschinen, bietet Copilot im Vergleich zu Autocomplete-Features ebenfalls erweiterte Funktionalität. Dazu zählen etwa das Kommentieren von Quellcode oder Implementierungsvorschläge ganzer Codeabschnitte. Neben den aktiv von Copilot unterbreiteten Vorschlägen ist es auch möglich, selbst umfangreichere Vervollständigungen des Werkzeugs anzufragen. Bei der Verwendung von Copilot ist es eine wichtige Fähigkeit zu lernen, mit welcher Arbeitsweise das Werkzeug bessere Vorschläge generiert. Dafür gibt es mehrere Ansätze — einer davon ist „Comment Driven“ zu arbeiten, wie unten vorgestellt — doch muss dies jeder Engineer für sich selbst und angepasst auf die eigene Arbeitsweise lernen.

Die folgenden Effekte sind im Umgang mit Copilot zu erwarten: Insgesamt erfährt die Qualität der Arbeit durch die Unterstützung von Copilot nur einen geringen Anstieg, wie in Unterabschnitt 4.2.3 diskutiert. Hauptprofiteure sind dabei mithilfe des Werkzeugs generierte Dokumentation und Kommentare. Die Codequalität verbessert sich nicht maßgeblich. Die Vorschläge müssen sorgfältig geprüft werden, um einen Abfall der Codequalität zu verhindern. Auch sind die Vorschläge von Copilot weniger nützlich in Codeabschnitten, die viel Domänenwissen beinhalten. Die Effizienz der Arbeit steigt ebenfalls. Insbesondere das Schreiben von Dokumentation und Kommentaren geht durch die Unterstützung von Copilot deutlich schneller. Auch das Schreiben repetitiver Codeabschnitte erfährt einen positiven Effekt. Beides wird, ähnlich wie bei ChatGPT, davon zurückgehalten, dass die Ausgaben von Copilot genau überprüft, überarbeitet oder verworfen werden müssen. Ein Fähigkeitszuwachs ist bei Copilot hingegen nicht zu erwarten. Zwar kann das Werkzeug auf low-level zu neuen Lösungen inspirieren, an die davor nicht gedacht wurde, aber dieser Effekt hält sich in Grenzen.

Auch der Umgang mit Copilot ist sehr komfortabel. Aufgrund der Integration in die IDE muss der eigene Workflow nicht verlassen werden. Im Vergleich zu ChatGPT fällt hier auch kein Zeitverlust durch die Übertragung an Kontextinformationen an. Darüber hinaus können unpassende Vorschläge auch ignoriert werden. Zurückgehalten wird der Komfort durch Usability Proble-

me, wie etwa Konflikte zwischen der Autovervollständigung der IDE und Copilot sowie durch fehlerhafte Vorschläge. So werden etwa non-existente Funktionen oder Bibliotheken halluziniert. Auch ist die Qualität der Vorschläge für komplexe Code-Abschnitte geringer.

5.5 BEST PRACTICES IM HINBLICK AUF ANWENDUNGSFÄLLE

Dieser Abschnitt präsentiert Best Practices, die sich auf konkrete Anwendungsfälle beziehen. Neben einem Anwendungsfall gehört zu einer dieser Best Practices ein Werkzeug — ChatGPT oder GitHub Copilot — welches das Software Engineering bereits heute effektiv unterstützen kann. Von allen 33 im Rahmen der Pilotuntersuchungen identifizierten Anwendungsfällen, sind nur zu denjenigen Fällen Best Practices aufgeführt, in welchen die Probanden ein Werkzeug gewinnbringend einsetzen konnten. Eine vollständige Übersicht über die rohen Anwendungsfälle und deren Auswertung ist in Abschnitt 4.3 zu finden. Besonders erfolgversprechende Best Practices sind als **klare Empfehlung** gekennzeichnet.

5.5.1 *Verstehen von Anforderungen*

Empfehlung

Bei dem Verstehen von technischen sowie fachlichen Anforderungen kann ChatGPT unterstützen. Aufgrund der enormen Menge von Trainingsdaten steht dem Chatbot ein umfassendes Weltwissen zur Verfügung, was ihn zu einer nützlichen Wissensressource macht. Der Chatbot ist hierbei kein Diskussionspartner auf Augenhöhe, sondern erweitert lediglich das verfügbare Wissen der Engineers. Mittels Fragen zu Voraussetzungen und Implikationen von Anforderungen hilft der Chatbot beim Brainstorming und dabei, keine Aspekte zu übersehen. Der Dialog mit ChatGPT ersetzt keinesfalls den Austausch mit Experten der Domäne, Kunden und anderen tatsächlichen Akteuren, sondern ergänzt diesen nur.

5.5.2 *Architekturpatterns & -Stile verstehen*

Klare Empfehlung

Beim Verstehen verschiedener Architekturpatterns und -Stile kann ChatGPT durch sein umfassendes Weltwissen helfen. Ähnlich wie bei der vorherigen Best Practice ist der Chatbot hier kein Diskussionspartner auf Augenhöhe, sondern dient lediglich als Wissensressource und Ergänzung zur Fachliteratur oder Onlinequellen. Ein Vorteil des Chatbots gegenüber herkömmlichen Quellen ist, dass er gezielt Fragen zu spezifischen Aspekten beantworten kann, ohne dass der Engineer große Informationsmengen händisch filtern und aufbereiten muss. Insbesondere im Vergleich zu Onlinequellen ist der Chatbot eine sinnvolle Alternative, welche keine Werbung oder Popups enthält und einen einheitlichen Sprachstil verwendet. Wie bei anderen Quellen auch sollte der Dialog mit ChatGPT jedoch nicht als einzige Quelle dienen. Die Ausgaben des Chatbots sind unter Umständen unvollständig, fehlerhaft oder beinhalten Hal-

luzinationen (siehe Unterabschnitt 2.4.3). Trotz dieser Einschränkung wurde die Best Practice in den Pilotuntersuchungen als besonders nützlich bewertet.

5.5.3 *Komponentenschnitt erarbeiten*

Empfehlung

Im Entwurf einer Lösungsarchitektur kann ChatGPT bei der Zerlegung eines Systems in Komponenten unterstützen. Eine Lösungsarchitektur ist nie richtig oder falsch, sondern muss immer differenziert abgewogen werden. ChatGPT kann dabei unterstützten, die Perspektive einer Engineer um Ideen und Inspirationen zu erweitern. Dabei kann der Chatbot nicht die Rolle eines Gesprächspartners auf Augenhöhe einnehmen, sondern lediglich Vorschläge zur Aufteilung unterbreiten. Auch bei dieser Best Practice ist ein großer Vorzug, dass ChatGPT diese Vorschläge auf Basis einer individuellen Situation machen kann. Um diese Situation zu beschreiben ist unter Umständen eine detaillierte Erklärung des Systems und der Rahmenbedingung notwendig. Das kann, ähnlich der vorhergegangenen Best Practice, viel Zeit in Anspruch nehmen und den Geheimnisschutz verletzen. Alles in allem scheint bei dieser Best Practice Potenzial für gewinnbringende Unterstützung durch ChatGPT zu bestehen, auch wenn er von nur wenigen Probanden getestet wurde.

5.5.4 *Trade-offs unterschiedlicher Lösungen vergleichen*

Empfehlung

Auch bei dem Vergleich von Trade-offs zwischen unterschiedlichen Lösungen und Designs kann ChatGPT aufgrund seines großen Weltwissens als Ressource eingesetzt werden. Zwar kann der Chatbot keine Architekturentscheidungen treffen, jedoch kann er die Engineers als Wissensressource unterstützen. Im Vergleich zu traditionellen Quellen wie Fachliteratur oder Onlinequellen leistet der Chatbot bei dieser Best Practice sogar mehr als diese Quellen. Obwohl ChatGPT nicht über tatsächliches Verständnis über Softwarearchitektur verfügt, kann er individuelle Lösungen miteinander in Bezug setzen und spezifische Constraints berücksichtigen, ohne dass diese in der Literatur explizit auftauchen. Allerdings ist es unter Umständen notwendig, dem Chatbot diese individuellen Lösungen und Constraints in einem hohen Detailgrad zu erklären, was viel Zeit in Anspruch nehmen kann. Besonders bei dieser Best Practice besteht die Gefahr, den Geheimnisschutz zu verletzen, da gegebenenfalls sensible Informationen über das System notwendig sind, um architekturelle Trade-offs vergleichen zu können.

5.5.5 *Schnittstellenschema erstellen*

Klare Empfehlung

Diese Best Practice fällt in das Aufgabengebiet „4.2 Feindesign erstellen“. Es geht um die Erstellung von Schnittstellenschemata (etwa im JSON-Format) auf Basis bereits vorhandener Datenformate. Hier kann ChatGPT zu einem großen Zeitersparnis beitragen. Nicht nur ist das Generieren von technik-nahem Code sowie Boilerplate Code eine große Stärke von ChatGPT, auch fällt ein

Hindernis für den Effizienzgewinn weg: Die an den Chatbot zu übermittelnden Kontextinformationen sind bereits in Form der Datenformate beschrieben. Das beschleunigt die Formulierung eines Prompts: gewünschtes Schema-Format angeben, bestehende Datenformate in den Prompt kopieren und die Constraints angeben. Selbstverständlich muss die Ausgabe des Chatbots stets genau überprüft werden. Auch der Geheimnisschutz läuft hier Gefahr, verletzt zu werden. Insgesamt wurde diese Best Practice trotzdem einstimmig von den Probanden als erfolgreich bewertet.

5.5.6 *Code schreiben*

Empfehlung

Das Schreiben von Code ist ein sehr breit gefächerter Anwendungsfall, bei welchem sowohl ChatGPT als auch GitHub Copilot unterstützen können. Der Chatbot kann auf Basis textueller Prompts Code generieren. Dieser Code wird eventuell Fehler oder Mängel aufweisen, kann aber als Ausgangspunkt dienen. Häufig kann der ChatGPT diese Fehler selbst korrigieren, wenn er dazu aufgefordert wird. Besonders für neue Features oder neue Projekte kann der Chatbot schnell einen ersten Entwurf generieren, insofern er detaillierte Anweisungen erhält. GitHub Copilot kann hingegen erst sinnvoll unterstützen, nachdem ein erster Schritt in der Implementierung getan wurde — dafür jedoch über einen ersten Entwurf hinaus. Während zwar kein Anstieg der Codequalität zu erwarten ist, kann Copilot doch die Effizienz der Arbeit steigern. Es besteht die Möglichkeit, dass sich die Werkzeuge gegenseitig ergänzen. Darüber kann auf Basis der Untersuchungsergebnisse jedoch keine Aussage getroffen werden.

In beiden Fällen ist die Qualität der Ausgaben durchwachsen, doch müssen die Ausgaben nicht perfekt sein, um einen Mehrwert zu bieten. Die Effizienzsteigerung überwiegt in den meisten Fällen den Zeitaufwand der Prüfung und Korrektur der Ausgaben. Insgesamt scheint der Mehrwert stark von den Rahmenbedingungen abhängig, unter denen die Werkzeuge eingesetzt werden. Letztendlich muss jeder Engineer selbst entscheiden, inwieweit die Werkzeuge für ihn einen Mehrwert für diese Best Practice bieten.

5.5.7 *Refactoring & Codeoptimierung*

Empfehlung

ChatGPT kann bei dem Refactoring oder der Optimierung von Code unterstützen. Dazu kann der Chatbot nach Anpassungsvorschlägen für bestehende Codeabschnitte gefragt werden. Die Vorschläge können dann als Inspiration und Ausgangspunkt für die tatsächlichen Anpassungen dienen. Den Geheimnisschutz zu wahren ist bei dieser Best Practice eine besondere Herausforderung, da es sich bei dem zu optimierenden Code um sensible Informationen handeln könnte. Auch bei dieser Best Practice ist der Mehrwert von ChatGPTs Unterstützung stark von der individuellen Situation abzuhängen. Insgesamt scheint jedoch Potenzial für den Einsatz des Chatbots zu bestehen.

5.5.8 *Debugging & Troubleshooting*

Empfehlung

Auch beim Identifizieren und Beheben von Fehlern kann ChatGPT unterstützen, indem der Chatbot nach Fehlerursachen zu gegebenen Codeabschnitten gefragt wird. Hierbei weist der Chatbot in die grobe Richtung einer korrekten Antwort, hilft jedoch nicht zwangsweise dabei, die Fehler auch zu beheben. Genau wie bei der vorherigen Best Practice „Refactoring & Codeoptimierung", stellt die Wahrung des Geheimnisschutzes eine besondere Herausforderung dar, da es sich bei dem fehlerhafen Code um sensible Informationen handeln könnte. Auch bei dieser Best Practice profitieren Engineers von dem enormen Weltwissen des Chatbots. Da dieses jedoch aufgrund der zeitbegrenzten Trainingsdaten unvollständig ist, könnte der Chatbot nicht über die notwendigen Informationen verfügen, um bei aktuellen oder neuen Fehlern zu helfen. Der Mehrwert der Unterstützung ist auch bei dieser Best Practice stark von der individuellen Situation abhängig.

5.5.9 *Fehlermeldungen analysieren*

Empfehlung

Bei dieser Best Practice handelt es sich um eine Spezialisierung der vorherigen Best Practice „Debugging & Troubleshooting". Hier kann ebenfalls ChatGPT dabei unterstützen, Fehlermeldungen auf den Grund zu gehen und auch hier hilft ChatGPT dabei in die Richtung einer korrekten Antwort zu weisen, weniger aber beim Beheben der Ursache. Der große Vorteil der Verwendung des Chatbots ist seine Schnelligkeit gegenüber herkömmlichen Quellen wie etwa der Websuche. Unter Verwendung von ChatGPT ist es nicht notwendig, eine große Menge an Informationen zu sichten und durchsuchen, wenn auch die Antworten des Chatbots unter Umständen weniger verlässlich sind. Diese Best Practice kann unkompliziert erprobt und schnell abgebrochen werden, sollte der Chatbot in der aktuellen Situation keinen Mehrwert liefern.

5.5.10 *Codeabschnitte verstehen*

Empfehlung

Beim Verstehen von Codeabschnitten können sowohl ChatGPT als auch Copilot unterstützen. ChatGPT kann dabei explizit nach Erklärungen zu Codeabschnitten gefragt werden, während Copilot Abschnitte erklären kann, indem eine Engineer oberhalb der entsprechenden Codezeilen einen Kommentar eröffnet und den Assistenten dessen Inhalt generieren lässt. In beiden Fällen ist die Einhaltung des Geheimnisschutzes ein Aspekt, der potenzielle Nutzung einschränken könnte.

5.5.11 *Lernen neuer Programmiersprachen*

Empfehlung

Beim Lernen neuer Programmiersprachen oder Frameworks kann ChatGPT helfen. Das gilt auch für neue Aspekte in bereits vertrauten Programmiersprachen. Der Chatbot kann Code generieren und diesen erklären, sowie Fragen

beantworten, die sonst Kollegen gestellt werden müssten. Ebenso kann der Chatbot durch die Unterstützung bei den oben genannten Best Practice „Debugging & Troubleshooting“, „Fehlermeldungen analysieren“ und „Codeabschnitte verstehen“ beim Lernen neuer Programmiersprachen helfen. Auch bei dieser Best Practice kann der individuelle Mehrwert variieren und stark von der individuellen Situation abhängen.

5.5.12 *Coding mit Domain-Specific Languages*

Klare Empfehlung

Diese Best Practice behandelt einen Spezialfall von „Code schreiben“, bei welchem ChatGPT sehr gewinnbringend eingesetzt werden kann. Bei domänenspezifischen Sprachen (DSL) handelt es sich um spezialisierte Programmiersprachen von begrenzter Ausdrucksmächtigkeit, die in spezifischen Domänen eingesetzt werden [Fow10, S. 27]. Dazu zählen etwa SQL oder reguläre Ausdrücke. Der Chatbot kann Fragen zu Syntax und Konzepten beantworten sowie Code in diesen Sprachen generieren. Unter Umständen können die Kompetenzen des Chatbots bei den vorherigen, Coding-bezogenen Best Practice auch der Arbeit mit DSLs zugute zu kommen. Die Unterstützung des Chatbots ist bei dieser Best Practice zudem deutlich konsistenter als bei den anderen Best Practices.

5.5.13 *Code kommentieren (inline)*

Klare Empfehlung

Das Kommentieren von Code wird im Kontext dieser Arbeit in zwei Ausprägungen betrachtet. „Inline“ bezeichnet hier Kommentare im Quellcode. In dieser Best Practice können sowohl ChatGPT als auch GitHub Copilot unterstützen, wobei nur die Unterstützung von Copilot eine klare Empfehlung ist. ChatGPT erstellt Kommentare als Teil des generieren Codes, benötigte hierzu allerdings eine detaillierte Beschreibung der gewünschten Ergebnisse. Auch individuellen Code kann der Chatbot kommentieren, insofern dieser in den Prompt kopiert wird. Copilot hingegen integriert sich nahtlos in den Workflow der Engineers und unterbreitet Vorschläge für Kommentare, sobald diese einen Kommentar eröffnen. Gerade im Umgang mit Copilot ist ein Geschwindigkeitszuwachs bei der Formulierung von Quellcode-Kommentaren zu erwarten.

5.5.14 *Code kommentieren (out-of-code)*

Empfehlung

Das Kommentieren von Code wird im Kontext dieser Arbeit in zwei Ausprägungen betrachtet. Dabei liegt „out-of-code“ Dokumentation separat vom Quellcode. Hierbei kann GitHub Copilot gewinnbringend eingesetzt werden. Ähnlich der vorherigen Best Practice ist aufgrund der guten Integration in den Workflow der Engineers ein Geschwindigkeitszuwachs bei der Formulierung von Dokumentation zu erwarten. Der Mehrwert fällt jedoch tendenziell weniger gut aus — womöglich aufgrund höherer fachlicher Komplexität und

höherer Distanz zwischen der Dokumentation und dem Quellcode, was eine Schwäche Copilots darstellt. Alles in allem leistet der Assistent hier trotzdem einen Mehrwert als Formulierungshilfe.

5.5.15 *Testdaten erstellen*

Klare Empfehlung

Bei der Erstellung von Testdaten können sowohl ChatGPT als auch GitHub Copilot unterstützen, wobei jedoch nur die Unterstützung von Copilot eine klare Empfehlung darstellt. ChatGPT kann zur Generierung aufgefordert werden, während Copilot die Testdaten als Teil von generiertem Code erstellt. Gerade im Umgang mit Copilot trägt das zu einem hohen Geschwindigkeitszuwachs bei, da kein langer Prompt erstellt werden muss. Eine Einschränkung dieser Best Practice, über die sich die Engineers bewusst sein müssen, ist Bias. Insbesondere bei der Erstellung von Testdaten kann Bias dazu führen, dass die erstellten Daten unter Umständen nicht repräsentativ sind und zu Klischees oder Allgemeinem tendieren und dazu, dass Edge-Cases nicht abgedeckt werden. Insofern diese Einschränkung berücksichtigt wird, kann Copilot hier einen großen Mehrwert bieten.

5.5.16 *Automatisierte Tests schreiben*

Empfehlung

Beim Schreiben von Testcode können ebenfalls die Werkzeuge unterstützen. Der Chatbot kann gebeten werden, Testcode für bestehenden Code zu schreiben, während Copilot ähnlich zur Best Practice „Code schreiben“ einfach beim Coding unterstützt. Beim Einsatz von ChatGPT kann dabei viel Zeit durch die Generierung des Testcodes gespart werden, da der zu testende Code (und damit ein relevanter Teil des zu übermittelnden Kontextes) bereits vorliegt — auch wenn der Geheimnisschutz hier eine Herausforderung darstellt. Copilot scheint für diese Best Practice ebenfalls gut geeignet zu sein, da Testcode oft repetitiv ist und dieser eine Stärke des Assistenten ist. In beiden Fällen können die Werkzeuge zu einem Geschwindigkeitsgewinn beitragen.

5.5.17 *Testcode kommentieren*

Empfehlung

Bei dieser Best Practice geht es um das „inline“-Kommentieren von Testcode. Hier kann GitHub Copilot unterstützen, indem der Assistent Vorschläge für Kommentare unterbreitet, sobald ein Engineer einen Kommentar beginnt. Im Gegensatz zur vorher aufgeführten Best Practice „Code kommentieren (inline)“ ist die Unterstützung von Copilot hier keine klare Empfehlung. Testcode beinhaltet im Vergleich zu „normalem“, Nicht-Test Code unter Umständen ein höheres Maß an Fachlichkeit, was eine Herausforderung für Copilot darstellen kann. Trotzdem kann der Assistent hier einen Mehrwert bieten, da Testcode oft repetitiv ist und dieser eine Stärke des Assistenten ist.

5.5.18 *Einrichten der Arbeitsumgebung*

Klare Empfehlung

Bei der Einrichtung und Konfiguration einer Arbeitsumgebung, um die Arbeit an einem Projekt zu ermöglichen, ist die Unterstützung von ChatGPT eine klare Empfehlung. Hierbei beantwortet der Chatbot Fragen zu Konfigurationen und Einstellungen, etwa von der Entwicklungsumgebung (IDE). Ähnlich wie bei der Best Practice „Fehlermeldungen analysieren" stellt der Chatbot hier eine Alternative zu herkömmlichen Quellen wie etwa der Websuche dar. Das kommt mit denselben Vor- und Nachteilen: Ein Engineer ist nicht gezwungen eine potenziell große Menge an Informationen zu sichten und zu bewerten, jedoch ist der Chatbot aufgrund seiner zeitlich begrenzten Trainingsdaten und Fehleranfälligkeit unter Umständen weniger verlässlich als eine Websuche. Auch diese Best Practice ist ähnlich leichtgewichtig, kann ohne viel Aufwand erprobt und schnell abgebrochen werden, insofern kein Mehrwert entsteht.

5.6 WEITERE BEST PRACTICES MIT CHATGPT

Die in Abschnitt 5.2 aufgestellten Einschränkungen und Hinweise sind bei der Arbeit mit ChatGPT zu beachten. Insbesondere die dort erwähnten Punkte *Kritische Prüfung der Ausgaben* des Chatbots, sowie *Sicherstellung von Daten- und Geheimnisschutz* könnten auch als Best Practices bezeichnet werden. Darüber hinaus ergeben sich aus den Untersuchungen die folgenden Best Practices.

5.6.1 *Prompt Engineering*

Prompt Engineering ist die Ausarbeitung und Optimierung von Eingaben — Prompts — für große Sprachmodelle (LLMs). Die Art und Weise, wie Prompts gestellt und formuliert werden, kann einen großen Einfluss auf die Qualität der Antworten haben. Das macht Prompt Engineering zu einem relevanten Wissensbaustein für die Arbeit mit ChatGPT. Die Formulierung von Prompts liegt nicht im Fokus dieser Arbeit, sodass nur wenige Hinweise diesbezüglich gegeben werden können. Wichtiger als eine genaue Anleitung ist ein Bewusstsein für die Relevanz von Prompt Engineering. Mit diesem Bewusstsein können Engineers selbstständig lernen und weiter experimentieren. Folgende Hinweise haben den Probanden im Umgang mit ChatGPT geholfen:

- **Der Prompt sollte alle relevanten Informationen beinhalten.** Darunter fallen etwa: Versionen der verwendeten Programmiersprachen, Frameworks, Bibliotheken und Umgebungen. Auch fachliche Constraints, implizite Annahmen, Naming Conventions oder Code-Stile sollten angegeben werden, insofern relevant. Werden diese Informationen angegeben, verfügt der Chatbot über mehr Kontext und tendiert dementsprechend zu passenderen Antworten.

- **ChatGPT um Kurzfassung bitten.** Wird der Chatbot gebeten, sich in seinen Antworten kurzzufassen, spart dieser an Erklärungen und Floskeln. Nicht nur führt dies zu übersichtlicheren Antworten und damit schnelleren Ausgaben, auch wird das Tokenlimit (siehe Unterabschnitt 5.6.2) effizienter genutzt.

5.6.2 *Bewusstsein für das Token Limit*

ChatGPT ist pro Anfrage auf eine bestimmte Anzahl sogenannter Tokens beschränkt. Ein Token entspricht dabei etwa ein vier Buchstaben (in der englischen Sprache). Das Token Limit ist ausschlaggebend für den Kontext, welchen der Chatbot je Anfrage berücksichtigen kann. Übersteigt der Chatverlauf das Token Limit, so wird der älteste Teil des Kontexts abgeschnitten. Das funktioniert ähnlich zu dem Sliding Window Prinzip: Werden bei erreichtem Token Limit 150 neue Tokens für einen weiteren Prompt und dessen Ausgabe benötigt, so werden die ältesten 150 Tokens abgeschnitten. Der Chatbot „vergisst“ gewissermaßen den Anfang des Chatverlaufes. Siehe Unterabschnitt 2.4.3 für eine detaillierte Beschreibung dieser Limitierung. Ebenfalls relevant ist, dass Quellcode besonders vielen Tokens entspricht. Gerade bei mehreren Iterationen von Quellcode innerhalb eines Chats kann das Token Limit schnell erreicht werden. Das macht es unter Umständen erforderlich, wichtige Informationen — wie etwa Kontext oder Anforderungen — zu wiederholen oder einen neuen Chat zu eröffnen. Im Fall eines neuen Chats müssen die wichtigen Informationen und die jüngste Iteration des Quellcodes dort erneut angegeben werden. Bewusstsein für diese Einschränkung hilft, die Grenzen des Chatbots besser zu verstehen.

5.6.3 *Kleinteilig arbeiten*

ChatGPT zu bitten, den Quellcode für eine ganze Applikation zu generieren, welche Bug-frei ist, alle Anforderungen erfüllt und dazu auch noch wirtschaftlich erfolgreich ist, ist selbstverständlich nicht sinnvoll. Deutlich erfolgreicher ist es hingegen, kleinteilig mit ChatGPT zu arbeiten — also nur einzelne Funktionen oder Codeabschnitte auf einmal zu generieren. Dadurch bleiben die Anweisungen sehr konkret und der Chatbot liefert bessere Ergebnisse. Außerdem lassen sich auf diese Weise die Ergebnisse leichter bewerten und verstehen.

5.6.4 *Eine Timebox setzen*

Bei der iterativen Arbeitsweise mit ChatGPT, bei welcher Prompts in mehreren Iterationen erarbeitet werden und viel mit dem Chatbot kommuniziert wird, kann es vorkommen, dass sich die Unterhaltung mit dem Chatbot im Kreis dreht, ohne bei einem Ergebnis näherzukommen. Das Setzen einer Timebox bei der Arbeit mit ChatGPT kann helfen, in solchen Fällen keine Zeit zu

verschwenden. Sobald die Timebox erreicht ist, kann auf alternative Quellen, Werkzeuge oder die Unterstützung von Kollegen zurückgegriffen werden.

5.6.5 *Experimentieren und Erfahrung sammeln*

Selbst ein Nutzungskonzept kann im Umgang mit einer neuen Technologie nur bedingt helfen. Große Sprachmodelle und darauf basierende Chatbots unterscheiden sich so signifikant von bisher verfügbaren Werkzeugen, dass die Sammlung eigener Erfahrung und das Experimentieren mit dem Werkzeug unerlässlich ist. Dabei ist wichtig, die Kompetenzen und Limitationen des Werkzeugs zu verstehen und einschätzen zu lernen — egal in welchem Anwendungsfall genau es eingesetzt wird.

5.7 WEITERE BEST PRACTICES MIT GITHUB COPILOT

Auch für die Arbeit mit GitHub Copilot sind die in Abschnitt 5.2 aufgestellten Einschränkungen und Hinweise zu beachten. Ebenfalls können insbesondere die dort erwähnten Punkte *Kritische Prüfung der Ausgaben* des Chatbots, sowie *Sicherstellung von Daten- und Geheimnisschutz* auch als Best Practices bezeichnet werden. Insgesamt fügt sich Copilot sehr gut in den Workflow der Engineers ein, sodass im Rahmen der Untersuchungen nur wenige Best Practices zustande gekommen sind.

5.7.1 *„Comment Driven" Programmieren*

Zum einen kann GitHub Copilot nur dann helfen, wenn ein erster Schritt bereits getan ist — etwa erste Zeilen Code geschrieben sind. Zum anderen scheitert der Assistent teilweise an komplexen Codeabschnitten oder wird durch fehlendes fachliches Verständnis zurückgehalten. Beiden dieser Probleme kann begegnet werden, indem „Comment Driven" gearbeitet wird. Dabei wird für die Implementierung eines Codeabschnitts zunächst ein Kommentar in Textform geschrieben, welcher die Funktionalität des zu implementierenden Codeabschnitts beschreibt. Anschließend wird der Cursor in die nächste Zeile bewegt und der Codeabschnitt mit Unterstützung von Copilot implementiert. Auf diese Weise hat Copilot mehr Kontext über die zu implementierende Funktionalität und scheint bessere Vorschläge zu unterbreiten. Auch kann Copilot so als Alternative zur Websuche verwendet werden. Ist beispielsweise unklar, wie mit der Python-Bibliothek *Pandas* ein *Dataframe*-Objekt aus einem Array an Objekten erstellt werden kann, ist es möglich, ein entsprechendes Kommentar (zum Beispiel # *Converting array to data frame*) zu schreiben, den Cursor in die nächste Zeile zu bewegen, und die von Copilot vorgeschlagene Umsetzung zu übernehmen. Die Anwendung dieser Best Practice führt jedoch nicht in allen Fällen einen Mehrwert. Auch hier müssen die Engineers auf Basis ihrer individuellen Situation entscheiden.

5.7.2 *Experimentieren und Erfahrung sammeln*

Auch ein Nutzungskonzept kann im Umgang mit einer neuen Technologie nur bedingt helfen. Große Sprachmodelle und auf diesen basierende KI-Pair-Progammer unterstützten das Software Engineering auf eine so signifikant andere Art als bisher verfügbare Werkzeuge, dass die Sammlung eigener Erfahrung und das Experimentieren mit dem Werkzeug elementar ist. Relevant ist dabei, die Kompetenzen und Limitationen des Werkzeugs zu kennenzulernen und zu verstehen — egal in welchem Anwendungsfall der Assistent konkret eingesetzt wird.

6 SCHLUSS

6.1 ZUSAMMENFASSUNG

Im Rahmen dieser Thesis wurde der Einsatz LLM-basierter Werkzeuge im Software Engineering untersucht und mittels einer Pilotuntersuchung praktisch erprobt. Zunächst wurde in Kapitel 2 auf die theoretischen Hintergründe der künstlichen Intelligenz, der Verarbeitung natürlicher Sprache, des maschinellen Lernens und großer Sprachmodelle eingegangen. Dabei standen die Kompetenzen und Limitationen großer Sprachmodelle besonders im Fokus. Diese Modelle verfügen aufgrund ihrer Komplexität über Emergent Abilities, durch welche sie vielseitig eingesetzt werden können und sogar Aufgaben bewältigen können, für welche sie nie explizit trainiert wurden. Gleichzeitig haben große Sprachmodelle auch erhebliche Limitationen, welche deren Einsatz in der Praxis einschränken — die signifikanteste Einschränkung ist, dass LLMs keine tatsächliche Intelligenz besitzen, sondern lediglich durch ihre Ausgaben den Eindruck erwecken, intelligent zu sein.

Anschließend wurde in Kapitel 3 eine Potenzialanalyse präsentiert, welche den Einsatz LLM-basierter Werkzeuge im Software Engineering von heute betrachtet, und dort aus zwei Perspektiven Potenziale identifiziert. Ansatz A betrachtet den Prozess des Software Engineerings nach der Darstellung von Accsos *BeST*-Foundation. Dort wurden die einzelnen Domänen des SE betrachtet und innerhalb dieser Domänen potenzielle Anwendungsfälle für den Einsatz der Werkzeuge erarbeitet, auf Basis der Kompetenzen und Limitationen der Technologie. Ansatz B evaluiert ein breites Kontingent an aktuell verfügbaren Werkzeugen auf deren Eignung für den Einsatz im Software Engineering sowie auf deren Eignung zur Erprobung in den Pilotuntersuchungen. Dort wurden die Werkzeuge entsprechend ihrer Art in vier Kategorien eingeteilt: „Intelligente“ Chatbots, KI-Pair-Programmer, verschiedene SE-Werkzeuge und Nicht-SE-Werkzeuge. Der Fokus der Untersuchungen wurde auf das gesamte Software Engineering gelegt, jedoch mit der Erwartung, dass die Domänen *3. Architektur und Design*, *4. Implementierung* und *5. Test* am meisten von der Unterstützung der Werkzeuge profitieren. Zur Erprobung wurde je ein Werkzeug aus den Kategorien „Intelligente“ Chatbots und KI-Pair-Programmer ausgewählt: ChatGPT und GitHub Copilot.

In Kapitel 4 wurden die Pilotuntersuchungen vorgestellt, in welchen die beiden Werkzeuge ChatGPT und GitHub Copilot auf ihre Eignung für den Einsatz im Software Engineering untersucht wurden. Dabei erprobte eine Gruppe aus 19 Testern — alle Software Engineers bei Accso — über einen Zeitraum von gut drei Monaten die Werkzeuge. Die Erfahrungen der Probanden wurden anschließend mittels Fragebogen und Experteninterviews erfasst und ausgewertet. Der Fragebogen umfasste Fragen zu Metainformationen, zu den

konkreten Erfahrungen und auch einordnende Fragen. Es wurden 33 Anwendungsfälle identifiziert, in welchen die Probanden die Werkzeuge erprobt hatten. Wie die Auswertung zeigt, können KI-Werkzeuge das Software Engineering bereits heute in vielen Anwendungsfällen unterstützen. Darüber hinaus erhöht die Unterstützung durch ChatGPT in einigen Fällen die Qualität der Arbeit, wären beide Werkzeuge grundsätzlich vor allem die Arbeitsgeschwindigkeit steigern können. Alles in allem scheint der, durch die Unterstützung entstandene, Mehrwert sich jedoch nicht in jedem Fall gleichermaßen einzustellen. Die Schlussfolgerung liegt nahe, dass dies in Abhängigkeit von der individuellen Situation der Engineers steht.

Kapitel 5 stellte die Ergebnisse der Untersuchungen in Form eines Nutzungskonzepts für den Einsatz beider Werkzeuge im Software Engineering vor. Das Nutzungskonzept umfasst eine Vielzahl von Best Practices — sowohl zu konkreten Anwendungsfällen, als auch allgemein zur Nutzung der Werkzeuge. Ebenso umfasst das Nutzungskonzept Einschränkungen, die im Umgang mit den Werkzeugen beachtet werden müssen. Unter den Best Practices sind einige als klare Empfehlungen gekennzeichnet. Diese zeichnen sich durch besonders positive Erfahrungen der Probanden in den Pilotuntersuchungen und einen besonders großen zu erwartenden Mehrwert aus. Sie sind in Abschnitt 6.2 abschließend zusammengefasst.

6.2 FAZIT

Diese Arbeit ist ein erster Vorstoß in unbekannte Gefilde. Dort steht sie als eine Orientierungshilfe für diejenigen, die LLM-basierte Werkzeuge in ihren Arbeitsalltag integrieren wollen. Dabei basieren die Ergebnisse auf den Erfahrungen von 19 Probanden, welche die Werkzeuge in realen Projekten erprobt haben. Trotz einiger Limitationen zeigen diese Ergebnisse: LLM-basierte Werkzeuge können bereits heute gewinnbringend im SE eingesetzt werden. Sowohl ChatGPT als auch GitHub Copilot können in vielen Anwendungsfällen die Arbeit der Engineers unterstützen und so die Qualität und Effizienz der Arbeit steigern. Dabei werden auch einige klare Empfehlungen ausgesprochen für Fälle, in welchen die Werkzeuge besonders nützlich sind. Klare Empfehlungen für den Einsatz von ChatGPT sind das Verstehen von Architekturpatterns & -Stile, das Erstellen von Schnittstellenschemata, das Coding mit Domain-Specific Language (DSL), sowie die Einrichtung der Arbeitsumgebung. Klare Empfehlungen für den Einsatz von GitHub Copilot sind das Kommentieren von Quellcode und das Erstellen von Testdaten. Es wird allen Lesern, die an der Entwicklung von Software beteiligt sind, wärmstens empfohlen, sich mit dem Nutzungskonzept auseinander zu setzen und darüber hinaus die Werkzeuge selbst zu erproben und eigene Erfahrungen zu sammeln. Gerade da der Mehrwert durch die Unterstützung der Werkzeuge kein Garant ist, sondern stark von der individuellen Situation abhängt, ist die eigene Erfahrung elementar. Wenn auch unklar ist, wie die weitere Entwicklung der Technologie aussehen wird, so scheint es doch unausweichlich, dass LLMs in Zukunft eine immer größere Rolle im SE spielen werden. Das macht es dringend not-

wendig zu lernen LLMs einzuordnen, zu lernen LLMs zu verstehen sowie zu lernen LLMs zu verwenden.

Mit Hinblick auf das größere Gesamtbild liegt nahe, dass sich die Ergebnisse des Einsatzes von ChatGPT und GitHub Copilot auch auf weitere Werkzeuge aus deren Kategorien, „Intelligente“ Chatbots und KI-Pair-Programmer, übertragen lassen. Genauso bleiben die dargestellten Kompetenzen und Limitationen großer Sprachmodelle in weiteren Domänen des Software Engineering relevant, die in den Pilotuntersuchungen nicht konkret erprobt wurden. Damit sind die Ergebnisse dieser Arbeit durchaus auch über die Grenzen der untersuchten Werkzeuge und Domänen hinaus relevant. Trotzdem bleibt auch viel Forschungspotenzial offen, wie im nachfolgenden Abschnitt beschrieben. Weitere Arbeiten könnten die Untersuchungssystematik dieser Arbeit auf eine größere Gruppe ausdehnen, um die hier vorgestellten Ergebnisse zu untermauern oder gegebenenfalls zu korrigieren. Die vorliegende Arbeit ist lediglich ein erster Überblick über das vielschichtige Potenzial von LLM-basierten Werkzeugen im Software Engineering.

6.3 AUSBLICK

Mit den Ergebnissen dieser Arbeit verfügt Accso über Einblicke in die Möglichkeiten und Limitationen des Einsatzes LLM-basierter Werkzeuge im Software Engineering. Das Nutzungskonzept gibt praktische Hinweise, wie Qualität und Effizienz der Arbeit im Software Engineering gesteigert werden können. Es wird empfohlen, dieses Wissen an die Engineers heranzutragen und sie zum Einsatz der Werkzeuge zu ermutigen.

Große Sprachmodelle sind ein hochaktives Forschungsfeld, sodass die Einzelheiten der Erkenntnisse dieser Arbeit schon bald überholt sein könnten. Neuerungen könnten den Weg für den Einsatz bestehender und neuer Werkzeuge in weiteren Anwendungsfällen öffnen oder aktuelle Limitationen und Einschränkungen abmindern. So etwa GitHub Copilot X, welches einen Chatbot direkt in die Entwicklungsumgebung integriert und somit einen signifikanten Nachteil von ChatGPT ausgleicht: die Notwendigkeit dem Chatbot in umfangreichen Prompts den Kontext der Problemstellung zu erklären. Es ist wichtig, dieses Thema aufmerksam zu verfolgen und neue Entwicklung fortlaufend zu evaluieren. Einige Beispiele für derartige Neuerungen sind die folgenden:

- Die Limitation des Daten- und Geheimnisschutzes könnte durch lokal gehostete Modelle gelöst werden. Lokal lauffähige Modelle mögen aktuell noch nicht so kompetent sein wie die Modelle der großen Anbieter, doch scheint sich diese Lücke zu schließen [PA23]. Insofern es für Organisationen oder gar Individuen möglich sein wird, mit den hier evaluierten Werkzeugen vergleichbare Chatbots und Assistenten selbst zu hosten, so könnten diese Werkzeuge auch in Umgebungen eingesetzt werden, in welchen die Verarbeitung von Daten durch Dritte problematisch ist.

- Die Größe des Kontext (auch Token Limit) schien eine signifikante Einschränkung für den Umgang mit den Werkzeugen zu sein. Auch hier scheint die aktuelle Entwicklung Fortschritte zu erzielen — erst im September 2023 stellten Xiong et al. in [Xio+23] eine Serie großer Sprachmodelle mit einer effektiven Kontextlänge von knapp über 32 Tausend Tokens vor. Diese Entwicklung könnte LLMs noch geeigneter für beispielsweise den Einsatz im Testen von Software machen.

- Ein spekulatives Szenario illustriert ferner, wohin die Entwicklung führen könnte. Denkbar wäre zum Beispiel, dass Quellcode irgendwann keine Kommentare mehr beinhalten muss, da LLM-basierte Werkzeuge diese ad-hoc (also erst zum Zeitpunkt des Lesens) generieren können. So könnte die Qualität der Kommentare mit der Zeit zunehmen und von dem Fortschritt und der steigenden „Intelligenz“ der Werkzeuge profitieren. Diese Funktionalität könnte in moderne Entwicklungsumgebungen integriert werden, ähnlich wie IDEs heute Hinweise zu der Verwendung von Funktionen dynamisch bereitstellen.

Ein effektiveres und effizienteres Software Engineering hat umfassende gesellschaftliche und marktwirtschaftliche Implikationen: Von wirtschaftlichem Wachstum über bessere Verfügbarkeit von Software in allen Teilen der Welt bis hin zur gesteigerten Wettbewerbsfähigkeit von Unternehmen, welche diese Werkzeuge gezielt einsetzen. So könnte etwa die breite Nutzung KI dem zunehmenden Arbeitskräftemangel entgegenwirken und die langfristige wirtschaftliche Leistungsfähigkeit erhalten oder steigern.

Es bestehen einige Aspekte, welche diese Arbeit nicht abschließend untersuchen konnte. Zukünftige Arbeiten könnten eine größer angelegte Untersuchung mit mehr Teilnehmern und mehr Werkzeugen durchführen oder aber anstatt in die Breite, in die Tiefe gehen und einzelne Aspekte genauer beleuchten. In beiden Ansätzen ist es möglich, auf Basis einer umfangreicheren Datenmenge mit einer statistisch aussagekräftigeren Methodik zu belastbareren Ergebnissen zu kommen. Ebenfalls spannend wäre die Untersuchung der kontroversesten Ergebnisse — dort wo die Meinungen der Probanden am weitesten auseinander gehen — um die Hintergründe und Faktoren zu untersuchen, welche diese Uneinigkeiten bedingen. Insbesondere die Untersuchung der gegenteiligen Erfahrungen bieten signifikantes Potenzial für weiteren Erkenntnisgewinn. Auch die Anwendungsfälle, welche gut bewertet, jedoch nur von einzelnen Probanden erprobt wurden, sollten eingehender untersucht werden. Diese bergen ebenfalls das Potenzial, das Software Engineering gewinnbringend zu unterstützen. Alle diese Ergebnisse würden einen Beitrag dazu leisten, LLM-basierte Werkzeuge noch präziser und noch effektiver einzusetzen. Dadurch könnten die Vorzüge besser genutzt werden, während gleichzeitig die Einschränkungen abgemildert werden können.

Teil II

APPENDIX

A

ANHANG

A.1 ERWÄHNUNGEN VON CHATGPT IN DER C'T

In den 6 Monaten nach der Veröffentlichung von ChatGPT am 30. November 2022 erschienen insgesamt 14 Ausgaben der renommierten Computerzeitschrift *c't*. In 12 davon fand ChatGPT Erwähnung. Teils sogar mehrfach und häufig sogar auf der Titelseite. Betrachtet wird der Zeitraum vom 30.11.2022 bis zum 30.05.2023. Die erste Ausgabe in diesem Zeitraum ist die 26. Ausgabe 2022 (erschienen am 03.12.2022), die letzte ist die 13. Ausgabe 2023 (erschienen am 20.05.2023). Bis auf die erste und die letzte Ausgabe enthielten die folgenden Ausgaben der *c't* mindestens einen Artikel zum Thema ChatGPT:

- In der 1. Ausgabe 2023 auf Seite 46 im Artikel *„Facettenreicher Gesprächspartner — Die Text-KI ChatGPT schreibt Fachtexte, Prosa, Gedichte und Programmcode"*[1]
- In der 2. Ausgabe 2023 auf Seite 3 im Artikel *„ChatGPT: Disruptives Potenzial"*[2]
- In der 3. Ausgabe 2023 auf Seite 32 im Artikel *„KI rüttelt uns hier wach — Interview: Wie ChatGPT die Lehre verändert"*[3]
- In der 4. Ausgabe 2023 auf Seite 12 im Artikel *„Das Ende der Trefferlisten? — Suchmaschinen beantworten Fragen mithilfe von KI"*[4]
- In der 5. Ausgabe 2023 auf Seite 56 im Artikel *„Der universelle Texter — Warum ChatGPT so fasziniert"*[5]
- In der 6. Ausgabe 2023 auf Seite 14 im Artikel *„Frei erfunden — Microsoft kombiniert seine Suchmaschine Bing mit einem ChatGPT-Nachfolger"*[6]
- In der 7. Ausgabe 2023 auf Seite 148 im Artikel *„KI rappt wie Eminem — Wie man mit Chatbots und Sprachtools einen Rap-Song produziert"*[7],
- In der 8. Ausgabe 2023 auf Seite 108 im Artikel *„HackGPT — ChatGPT als Hacking-Tool"*[8],

1 https://www.heise.de/select/ct/2023/1/2233908274346530870; Aufgerufen am 30.05.2023
2 https://www.heise.de/select/ct/2023/2/2230011003384627303; Aufgerufen am 30.05.2023
3 https://www.heise.de/select/ct/2023/3/2234607435104665729; Aufgerufen am 30.05.2023
4 https://www.heise.de/select/ct/2023/4/2300610555933424240; Aufgerufen am 30.05.2023
5 https://www.heise.de/select/ct/2023/5/2301615471609290458; Aufgerufen am 30.05.2023
6 https://www.heise.de/select/ct/2023/6/2303314493063464861; Aufgerufen am 30.05.2023
7 https://www.heise.de/select/ct/2023/7/2304610053828811986; Aufgerufen am 30.05.2023
8 https://www.heise.de/select/ct/2023/8/2305815052110670001; Aufgerufen am 30.05.2023

- In der 9. Ausgabe 2023 auf Seite 126 im Artikel *„Schneller als gedacht — ChatGPT zwischen wirtschaftlicher Effizienz und menschlichem Wunschdenken“*[9],
- In der 10. Ausgabe 2023 auf Seite 35 im Artikel *„ChatGPT findet Malware in Softwarepaketen“*[10],
- In der 11. Ausgabe 2023 auf Seite 146 im Artikel *„…mit Sprach-KI im Tandem — Wie ChatGPT beim Programmieren hilft“*[11],
- In der 12. Ausgabe 2023 auf Seite 78 im Artikel *„Die c't in Zeiten der künstlichen Intelligenz“*[12],

9 https://www.heise.de/select/ct/2023/9/2304715244186570960; Aufgerufen am 30.05.2023
10 https://www.heise.de/select/ct/2023/10/2309009342566069855; Aufgerufen am 30.05.2023
11 https://www.heise.de/select/ct/2023/11/2310109593910076813; Aufgerufen am 30.05.2023
12 https://www.heise.de/select/ct/2023/12/2306013140164748666; Aufgerufen am 30.05.2023

A.2 VOLLSTÄNDIGE ANTWORTEN AUF FRAGE 2.C

Tabelle A.1 gibt Übersicht über alle mehrfach verwendeten Programmiersprachen, Frameworks und Entwicklungsumgebungen im Kontext von Frage 2 c. Darüber hinaus gab es im Zusammenhang mit ChatGPT die folgenden Einzelnennungen: X-Code, Swift, Powershell, Assembler, GWT (Java Framework), Java Spring, Junit (Java), Mockito (Java), Pandas (Python), RegEx, Lua, JSON, Oracle Forms, PL/SQL, Unity Game Engine, Firebase, Kendo-UI, Mapstruct, Spring Cloud Contract, PostgreSQL, Liquibase, Hibernate, Svelte, NestJS, Next-JS, Firefox Webbrowser Plugin. Im Zusammenhang mit GitHub Copilot gab es die folgenden Einzelnennungen: JetBrains PyCharm, YAML, CDK (JSON-basierte Infrastruktur-Definition für AWS), RxJs, Jasmine (Test Suite), Karma (Test Suite), Playwright, .NET.

Programmiersprache, Framework oder Entwicklungsumgebung	ChatGPT	GitHub Copilot	Insgesamt
Typescript	6	8	14
Angular	6	5	11
Visual Studio Code	3	8	11
Java	7	2	9
C#	4	4	8
HTML	4	2	6
Python	3	2	5
JetBrains IntelliJ	3	2	5
Vue.js	3	2	5
Java Spring Boot	4	0	4
Javascript	3	1	4
JetBrains Rider	0	3	3
REST API	3	0	3
CSS	3	0	3
NgRX	1	1	2
ASP.NET Entity Framework Core	1	1	2
XML	2	0	2
Maven	2	0	2
SQL	2	0	2
Go	1	1	2
SCSS	1	1	2
Ascii-Doc	0	2	2

Tabelle A.1: Die Anzahl Nennungen aller mehrfach angegebenen Programmiersprachen, Frameworks und Entwicklungsumgebungen im Kontext von Frage 2 c. Dargestellt je KI-Werkzeug und insgesamte Nennungen.

A.3 STREUDIAGRAMME

In Abbildung A.1 ist eine Auswahl der Streudiagramme ausgeführt, welche nicht bereits in Unterabschnitt 4.4.1 gezeigt sind.

A.4 RÜCKFRAGEN — ENGLISCHE VERSION

In diesem Abschnitt hängen die in Unterabschnitt 4.4.4 vorgestellten Rückfragen in ihrer englischen Übersetzung an.

> Rückfrage 1:
> When interacting with ChatGPT, an issue that may arise is that the conversation with the chatbot spins in circles without getting any closer to a result. **Based on your experience** in working with **ChatGPT**, would you recommend setting a timebox when working with the chatbot to prevent wasting a lot of time in such cases?
>
> Rückfrage 2:
> GitHub Copilot tends to provide less accurate suggestions for more complex code sections. Additionally, Copilot can only assist once an initial step has been made in the implementation (such as writing the first lines of code). These challenges could possibly be addressed by using a „Comment Driven“ approach. In this method, a comment is first written in text form describing the functionality to be implemented for a code section. The cursor is then moved to the next line, and the code section is implemented using Copilot. This provides Copilot with more context.
>
> **Based on your experience** with **Copilot**, would you recommend using a „Comment Driven“ approach?

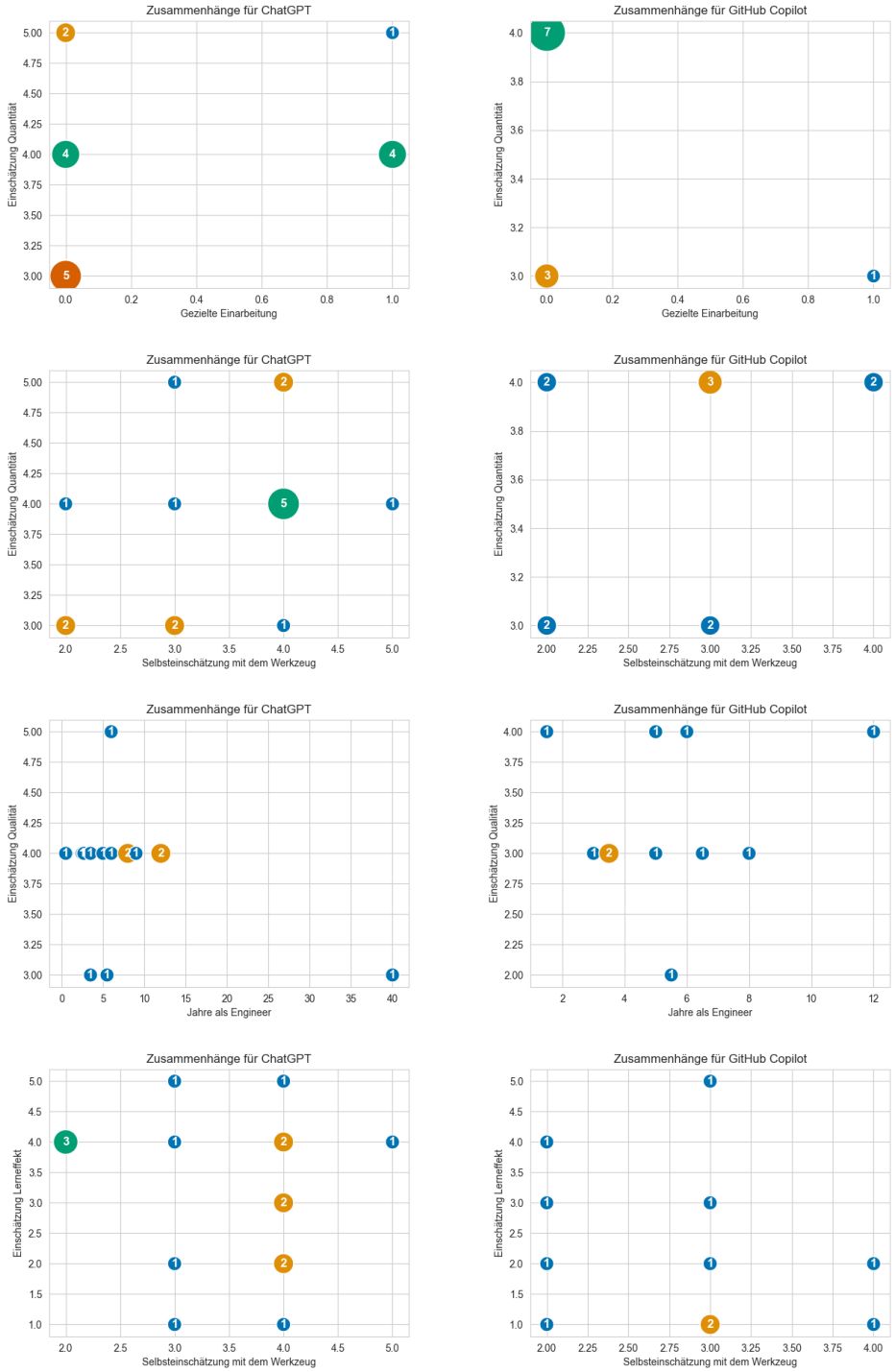


Abbildung A.1: Weitere Streudiagramme, welche die Kontext-Parameter auf die einordnenden Fragen 6 bis 8 abbilden.

A.5 FRAGEBOGEN: AI-ASSISTED SOFTWARE ENGINEERING — ENGLISCHE VARIANTE

Date: _____
Questionee: _____
Interviewer: _____

AI-Tools: _____: _____
_____: _____
_____: _____
_____: _____

Note: For the tools, please provide all details, if available (model, plugins used, ...).

1. **About you:**

   How long have you been working as a software engineer?

2. **Experience Level:**

   AI tool _____:
   I have been using the AI tool for *less than 1 week / less than 1 month / 1 to 3 months / more than 3 months*.

   I consider myself *very inexperienced / inexperienced / somewhat experienced / experienced / very experienced* in using the AI tool.

   Which programming languages, IDEs and/or frameworks have you used with this tool?

   In what context? (Client project, personal project, ...) How long have you been active in this context?

   AI tool _____:
   I have been using the AI tool for *less than 1 week / less than 1 month / 1 to 3 months / more than 3 months*.

   I consider myself *very inexperienced / inexperienced / somewhat experienced / experienced / very experienced* in using the AI tool.

Which programming languages, IDEs and/or frameworks have you used with this tool?

In what context? (Client project, personal project, ...) How long have you been active in this context?

AI tool _____:
I have been using the AI tool for *less than 1 week* / *less than 1 month* / *1 to 3 months* / *more than 3 months*.

I consider myself *very inexperienced* / *inexperienced* / *somewhat experienced* / *experienced* / *very experienced* in using the AI tool.

Which programming languages, IDEs and/or frameworks have you used with this tool?

In what context? (Client project, personal project, ...) How long have you been active in this context?

3. **Onboarding:**
   Did you deliberately look into how to use the AI tool? What did your introduction look like? How did you learn to work with the tool?

AI tool _____:

AI tool _____:

AI tool _____:

4. **Use Cases/Application Areas:**

AI tool _____:
Where and how did you use the tool? Where was it successful/unsuccessful? Why?

AI tool _____:
Where and how did you use the tool? Where was it successful/unsuccessful? Why?

AI tool _____:
Where and how did you use the tool? Where was it successful/unsuccessful? Why?

5. **Task Areas according to *BeST*:**
   Abbildung A.2 shows the *BeST-Foundation Framework*. Where can your experiences be classified in that context?

6. **Quality:**

AI tool _____:
By using the AI tool, the quality of my work is *significantly worse* / *worse* / *unchanged* / *higher* / *significantly higher*. (Quality can mean many things: code

quality, more comprehensive requirement consideration, ...)

Which tasks are affected? Why?

AI tool _____:
By using the AI tool, the quality of my work is *significantly worse* / *worse* / *unchanged* / *higher* / *significantly higher*. (Quality can mean many things: code quality, more comprehensive requirement consideration, ...)

Which tasks are affected? Why?

AI tool _____:
By using the AI tool, the quality of my work is *significantly worse* / *worse* / *unchanged* / *higher* / *significantly higher*. (Quality can mean many things: code quality, more comprehensive requirement consideration, ...)

Which tasks are affected? Why?

7. **Efficiency:**

AI tool _____:
With the support of the AI tool, I worked *significantly slower* / *slower* / *at unchanged speed* / *faster* / *significantly faster*.

Which tasks are affected? Why?

AI tool _____:
With the support of the AI tool, I worked *significantly slower* / *slower* / *at unchanged speed* / *faster* / *significantly faster*.

Which tasks are affected? Why?

AI tool _____:
With the support of the AI tool, I worked *significantly slower / slower / at unchanged speed / faster / significantly faster*.

Which tasks are affected? Why?

8. **Skills/Learning Effect:**

AI tool _____:
The AI tool allows me to do things that I could not do without it. I *strongly disagree / disagree / am neutral / agree / strongly agree*.

Which skills are affected? Why?

AI tool _____:
The AI tool allows me to do things that I could not do without it. I *strongly disagree / disagree / am neutral / agree / strongly agree*.

Which skills are affected? Why?

AI tool _____:
The AI tool allows me to do things that I could not do without it. I *strongly disagree / disagree / am neutral / agree / strongly agree*.

Which skills are affected? Why?

9. **Potential:**

AI tool _____:
At which tasks do you see potential for the AI tool? Based on your experience, which tasks seem within reach of the tool in the future?

AI tool _____:
At which tasks do you see potential for the AI tool? Based on your experience, which tasks seem within reach of the tool in the future?

AI tool _____:
At which tasks do you see potential for the AI tool? Based on your experience, which tasks seem within reach of the tool in the future?

10. **Satisfaction & Well-being:**

AI tool _____:
I will *definitely not* / *not* / *maybe* / *continue to* / *definitely continue to* use the AI tool in my daily SE routine.

How did you find working with these tools? Comfortable? Exhausting? Comparison of tools with each other?

AI tool _____:
I will *definitely not* / *not* / *maybe* / *continue to* / *definitely continue to* use the AI tool in my daily SE routine.

How did you find working with these tools? Comfortable? Exhausting? Comparison of tools with each other?

AI tool _____:
I will *definitely not* / *not* / *maybe* / *continue to* / *definitely continue to* use the AI tool in my daily SE routine.

How did you find working with these tools? Comfortable? Exhausting? Comparison of tools with each other?

11. **Negative Aspects:**

AI tool _____:
Where, in your eyes, has the tool failed? What did you not like?

AI tool _____:
Where, in your eyes, has the tool failed? What did you not like?

AI tool _____:
Where, in your eyes, has the tool failed? What did you not like?

12. **Lessons Learned:**

AI tool _____:
What tips & advice would you give to someone who is using the tool for the first time?

AI tool _____:
What tips & advice would you give to someone who is using the tool for the first time?

AI tool _____:
What tips & advice would you give to someone who is using the tool for the first time?

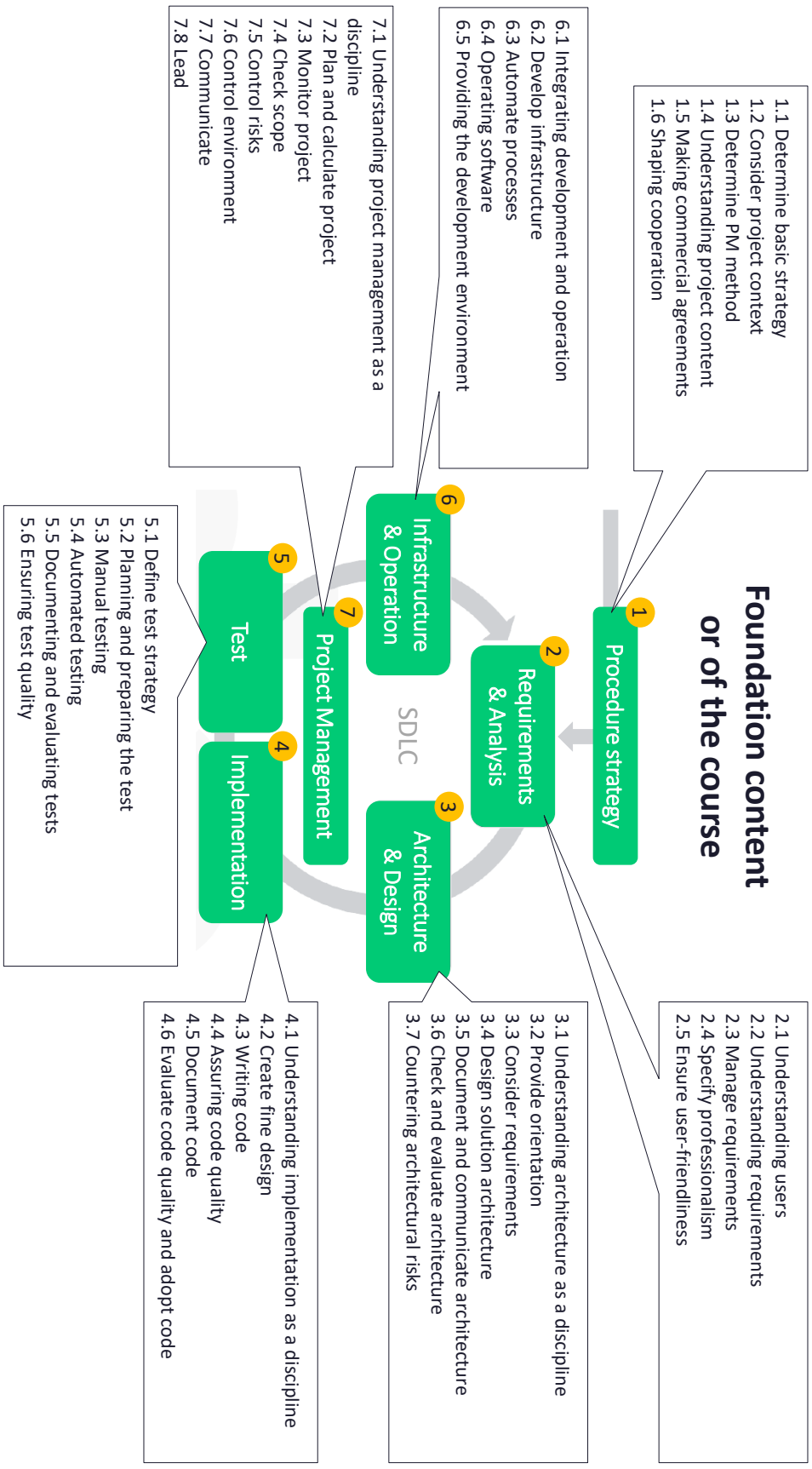


Abbildung A.2: Overview of the software development process. The scale of the rating is: *significantly unsuccessful* (1) / *unsuccessful* (2) / *mixed* (3) / *successful* (4) / *significantly successful* (5).

[Ahm+23] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Mst Shamima Aktar und Tommi Mikkonen. "Towards human-bot collaborative software architecting with chatgpt". In: *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 2023, S. 279–285.

[Ben+21] Emily M Bender, Timnit Gebru, Angelina McMillan-Major und Shmargaret Shmitchell. "On the dangers of stochastic parrots: Can language models be too big?" In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 2021, S. 610–623.

[Bro+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell u. a. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), S. 1877–1901.

[Bry19] Marc Brysbaert. "How many words do we read per minute? A review and meta-analysis of reading rate". In: *Journal of memory and language* 109 (2019), S. 104047.

[CZZ23] Lingjiao Chen, Matei Zaharia und James Zou. "How is ChatGPT's behavior changing over time?" In: *arXiv preprint arXiv:2307.09009* (2023).

[Cho20] KR Chowdhary. *Fundamentals of artificial intelligence*. Springer, 2020.

[CCC23] Orges Cico, Betim Cico und Andja Cico. "AI-assisted Software Engineering: a tertiary study". In: *2023 12th Mediterranean Conference on Embedded Computing (MECO)*. IEEE. 2023, S. 1–6.

[Cop] B.J. Copeland. *artificial intelligence*. URL: https://www.britannica.com/technology/artificial-intelligence (besucht am 21. 05. 2023).

[Dav23] Emilia David. *George R.R. Martin and other authors sue OpenAI for copyright infringement*. The Verge. 20. Sep. 2023. URL: https://www.theverge.com/2023/9/20/23882140/george-r-r-martin-lawsuit-openai-copyright-infringement (besucht am 08. 10. 2023).

[Del+23] Fabrizio Dell'Acqua, Edward McFowland, Ethan R Mollick, Hila Lifshitz-Assaf, Katherine Kellogg, Saran Rajendran, Lisa Krayer, François Candelon und Karim R Lakhani. "Navigating the Jagged Technological Frontier: Field Experimental Evidence of

the Effects of AI on Knowledge Worker Productivity and Quality". In: *Harvard Business School Technology & Operations Mgt. Unit Working Paper* 24-013 (2023).

[Die18] Gabriele Diewald. "Zur Diskussion: Geschlechtergerechte Sprache als Thema der germanistischen Linguistik–exemplarisch exerziert am Streit um das sogenannte generische Maskulinum". In: *Zeitschrift für germanistische Linguistik* 46.2 (2018), S. 283–299.

[Don23] Andreas Donath. *Geheimnisverrat: Amazon warnt Mitarbeiter vor ChatGPT*. 2023. URL: https://www.golem.de/news/geheimnisverrat-amazon-warnt-mitarbeiter-vor-chatgpt-2302-171975.html (besucht am 20. 06. 2023).

[Enn] Christiana Ennemoser. *heute journal interviewt KI: „Immense Vorteile, die genutzt werden können "*. ZDF. URL: https://www.zdf.de/nachrichten/digitales/ki-interview-chatgpt-heute-journal-100.html (besucht am 23. 05. 2023).

[Fow10] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

[Fra20] Alexander L Fradkov. "Early history of machine learning". In: *IFAC-PapersOnLine* 53.2 (2020), S. 1385–1390.

[Hum22] Bernhard G Humm. *Applied Artificial Intelligence; An Engineering Approach*. https://leanpub.com/AAI. Leanpub, 2022.

[Jan23] Matthias Janson. *ChatGPT's Sprint zu einer Million Nutzer:innen*. Statista, 2023. URL: https://de.statista.com/infografik/29195/zeitraum-den-online-dienste-gebraucht-haben-um-eine-million-nutzer-zu-erreichen (besucht am 30. 05. 2023).

[Jon94] Karen Sparck Jones. "Natural language processing: a historical review". In: *Current issues in computational linguistics: in honour of Don Walker* (1994), S. 3–16.

[Kal22] Eirini Kalliamvakou. *Research: quantifying GitHub Copilot's impact on developer productivity and happiness*. 2022. URL: https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/ (besucht am 27. 09. 2023).

[Kay23] Grace Kay. *Bill Gates calls ChatGPT 'every bit as important as the PC' or the internet*. Business Insider. 2. Feb. 2023. URL: https://www.businessinsider.com/bill-gates-chatgpt-ai-artificial-intelligenct-as-important-pc-internet-2023-2 (besucht am 16. 05. 2023).

[Kip23] Roland Kipke. "Sind wir moralisch verpflichtet, eine gendergerechte Sprache zu verwenden?" In: *Zeitschrift für Ethik und Moralphilosophie* (2023), S. 1–22.

[Kro08] Anders Krogh. "What are artificial neural networks?" In: *Nature biotechnology* 26.2 (2008), S. 195–197.

[Mdr] *Künstliche Intelligenz Chat GPT - was ist das? Alle Fragen und Antworten rund um den Hype*. Mitteldeutscher Rundfunk. URL: https://www.mdr.de/brisant/ratgeber/chat-gpt-178.html (besucht am 23. 05. 2023).

[Goo] *Large Language Models in „Software Engineering“ — Google Scholar*. 10. Okt. 2023. URL: https://scholar.google.com/scholar?hl=de&as_sdt=0%2C5&as_ylo=2023&q=Large+Language+Models+in+%22Software+Engineering%22&btnG= (besucht am 10. 10. 2023).

[Lin23] Michael Linden. *Geschäftsgeheimnisse: Apple verbietet Nutzung von ChatGPT und Github Copilot*. 2023. URL: https://www.golem.de/news/geschaeftsgeheimnisse-apple-verbietet-nutzung-von-chatgpt-und-github-copilot-2305-174285.html (besucht am 20. 06. 2023).

[Mau23] Cecily Mauran. *OpenAI is being sued for training ChatGPT with 'stolen' personal data*. Mashable. 29. Juni 2023. URL: https://mashable.com/article/openai-chatgpt-class-action-lawsuit (besucht am 08. 10. 2023).

[MY15] Iqbal Muhammad und Zhu Yan. “SUPERVISED MACHINE LEARNING APPROACHES: A SURVEY.” In: *ICTACT Journal on Soft Computing* 5.3 (2015).

[MS21] Carolin Müller-Spitzer. “Geschlechtergerechte Sprache: Zumutung, Herausforderung, Notwendigkeit?” In: *Sprachreport* 37.2 (2021), S. 1–12.

[Nie13] Lene Nielsen. *Personas-user focused design*. Bd. 15. Springer, 2013.

[Nie15] Michael A Nielsen. *Neural networks and deep learning*. Bd. 25. Determination press San Francisco, CA, USA, 2015.

[Nor12] Peter Norvig. *English Letter Frequency Counts: Mayzner Revisited*. 2012. URL: http://norvig.com/mayzner.html (besucht am 25. 09. 2023).

[Ope22] OpenAI. *OpenAI Blog: Aligning language models to follow instructions*. 2022. URL: https://openai.com/research/instruction-following (besucht am 21. 09. 2023).

[Ope23a] OpenAI. *OpenAI Blog: GPT-4*. 2023. URL: https://openai.com/research/gpt-4 (besucht am 25. 09. 2023).

[Ope23b] OpenAI. *OpenAI Documentation: Models*. 2023. URL: https://platform.openai.com/docs/models (besucht am 25. 09. 2023).

[Ope23c] OpenAI. *Tokenizer*. https://platform.openai.com/tokenizer. [Online; zuletzt aufgerufen 10. September 2023]. 2023.

[Ope23d] OpenAI. *What are tokens and how to count them?* https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them. [Online; zuletzt aufgerufen 02. Oktober 2023]. 2023.

[Ouy+22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray u. a. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), S. 27730–27744.

[PA23] Dylan Patel und Afzal Ahmad. *Google: „We Have No Moat, And Neither Does OpenAI"*. https://www.semianalysis.com/p/google-we-have-no-moat-and-neither. [Online; zuletzt aufgerufen 04. Oktober 2023]. 2023.

[Pet22] Uwe Peters. "What is the function of confirmation bias?" In: *Erkenntnis* 87.3 (2022), S. 1351–1376.

[PFCL12] Jennifer L Prewitt-Freilino, T Andrew Caswell und Emmi K Laakso. "The gendering of language: A comparison of gender equality in countries with gendered, natural gender, and genderless languages". In: *Sex roles* 66.3 (2012), S. 268–281.

[Rad+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever u. a. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), S. 9.

[Ros+23] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller und Justin D Weisz. "The programmer's assistant: Conversational interaction with a large language model for software development". In: *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 2023, S. 491–514.

[Sha48] Claude E Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), S. 379–423.

[Sol] Accso Accelerated Solutions. *Die Accso Story — Spezialist:innen für anspuchsvolle, individuelle IT-Lösungen*. URL: https://accso.de/ueber-uns (besucht am 05. 10. 2023).

[Sol23] Accso Accelerated Solutions. *Foliensatz: Foundation für eine Beschleunigte Softwaretechnik*. Version 0.1.2. 2023.

[TPK20] Damian A Tamburri, Fabio Palomba und Rick Kazman. "Success and failure in software engineering: A followup systematic literature review". In: *IEEE Transactions on Engineering Management* 68.2 (2020), S. 599–611.

[TR+23] Archana Tikayat Ray, Bjorn F Cole, Olivia J Pinon Fischer, Anirudh Prabhakara Bhat, Ryan T White und Dimitri N Mavris. "Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models". In: *Systems* 11.7 (2023), S. 352.

[Tou+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar u. a. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser und Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Wan+23] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan und Anima Anandkumar. "Voyager: An open-ended embodied agent with large language models". In: *arXiv preprint arXiv:2305.16291* (2023).

[Wik23a] Wikipedia. *Ausschreibung — Wikipedia, die freie Enzyklopädie*. [Online; Stand 9. September 2023]. 2023. URL: \url{https://de.wikipedia.org/w/index.php?title=Ausschreibung&oldid=236402571}.

[Wik23b] Wikipedia. *C't — Wikipedia, die freie Enzyklopädie*. 2023. URL: https://de.wikipedia.org/w/index.php?title=C%E2%80%99t&oldid=233458094 (besucht am 30. 05. 2023).

[Wik23c] Wikipedia. *Schachtürke — Wikipedia, die freie Enzyklopädie*. [Online; Stand 16. September 2023]. 2023. URL: https://de.wikipedia.org/w/index.php?title=Schacht%C3%BCrke&oldid=233390146.

[Xio+23] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz u. a. "Effective Long-Context Scaling of Foundation Models". In: *arXiv preprint arXiv:2309.16039* (2023).

[Yan+23] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin und Xia Hu. "Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond". In: (2023). arXiv: 2304.13712 [cs.CL].

[Yet+23] Burak Yetiştiren, Işık Özsoy, Miray Ayerdem und Eray Tüzün. "Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT". In: *arXiv preprint arXiv:2304.10778* (2023).

[Zha+] Jianzhang Zhang, Yiyang Chen, Nan Niu und Chuang Liu. "Empirical Evaluation of Chatgpt on Requirements Information Retrieval Under Zero-Shot Setting". In: *Available at SSRN 4450322* ().

[Zha+23] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong u. a. "A survey of large language models". In: *arXiv preprint arXiv:2303.18223* (2023).

[Zie+22] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam und Edward Aftandilian. "Productivity assessment of neural code completion". In: *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 2022, S. 21–29.