

Bachelor Thesis

Die Zukunft der Software-Entwicklung unter dem Einfluss künstlicher Intelligenz: Chancen, Herausforderungen und praxisorientierte Anwendungen

zur Erlangung des akademischen Grades

Bachelor of Science

eingereicht im Fachbereich Mathematik, Naturwissenschaften und Informatik an der
Technischen Hochschule Mittelhessen

von

Jan Ole Schmidt

7. Juli 2025

Referent: Prof. Dr. Dennis Priefer

Korreferent: Kevin Linne

Erklärung zur Verwendung von generativer KI

In Übereinstimmung mit den Empfehlungen der Deutschen Forschungsgemeinschaft (DFG)¹ und denen der Zeitschrift Theoretical Computer Science² erkläre ich (der Autor/die Autorin) hiermit den Einsatz von generativer KI.

Bei der Vorbereitung dieser Arbeit habe ich ChatGPT 4 verwendet, um ausschließlich die Lesbarkeit und Sprache zu verbessern. Nach der Verwendung von ChatGPT 4 habe ich den Inhalt überprüft und nach Bedarf bearbeitet und übernehme die volle Verantwortung für den Inhalt dieser Arbeit.

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Gießen, den 7. Juli 2025

Jan Ole Schmidt

1 DFG formuliert Richtlinien für den Umgang mit generativen Modellen für Text- und Bild: <https://www.dfg.de/en/news/news-topics/announcements-proposals/2023/info-wissenschaft-23-72>

2 Erklärung zur Verwendung von generativer KI in wissenschaftlichen Arbeiten: <https://www.sciedirect.com/journal/theoretical-computer-science/publish/guide-for-authors>

Inhaltsverzeichnis

1 Einführung	2
1.1 Motivation	2
1.2 Zielsetzung und Fragestellungen	4
1.3 Methodik	4
1.4 Abgrenzung	5
2 Theoretische Grundlagen	7
2.1 Einordnung generativer KI in die Softwareentwicklung	7
2.1.1 Beispiele für generative KI-Tools in der Praxis	8
2.1.2 Wichtige Algorithmen und Modelle in der Softwareentwicklung .	10
2.2 Generative KI-Tools: Funktion und Anwendung	11
3 Praktische Demonstration	14
3.1 Zielsetzung und Vorgehen	14
3.2 Vorstellung der App „Locals“	15
3.3 Implementierung der interaktiven Kartenansicht mit KI-Unterstützung .	16
3.3.1 Prompt-Setup: Einheitliche Aufgabenstellung für alle Tools . . .	16
3.3.2 Demonstration mit GitHub Copilot	18
3.3.3 Demonstration mit Cursor	22
3.3.4 Demonstration mit Bolt	26
3.3.5 Vergleich und Bewertung der eingesetzten KI-Tools	29
3.3.6 Zwischenfazit	30
4 Chancen	32
4.1 Effizienzsteigerung und Automatisierung	32
4.2 Neue Werkzeuge und Methoden	34
5 Herausforderungen durch KI in der Softwareentwicklung	37
5.1 Sicherheits- und Datenschutzaspekte	37
5.1.1 Sicherheitsrisiken durch generative Modelle	38
5.2 Ethische und soziale Implikationen	38
5.2.1 Ethische Konflikte und Bias in KI-Systemen	39
5.2.2 Langfristige Auswirkungen auf Entwickler:innen-Rollen	40
5.2.3 Technische und organisatorische Hürden bei der Einführung von KI	40

6 Wirtschaftliche und gesellschaftliche Auswirkungen	42
6.1 Veränderungen in Softwareunternehmen	42
6.2 Auswirkungen auf den Arbeitsmarkt und Entwickler:innen-Rollen	43
6.3 Zukunftsperspektiven und strategische Empfehlungen	44
6.4 Kosten-Nutzen-Analyse von KI-gestützter Softwareentwicklung	45
6.5 Fazit	46
7 Fazit und Ausblick	47
7.1 Zusammenfassung der Kernergebnisse	47
7.2 Beantwortung der Forschungsfragen	48
7.3 Handlungsempfehlungen für Praxis und Unternehmen	49
7.4 Offene Fragen und Forschungsbedarf	50
7.5 Ausblick	51
Literaturverzeichnis	52
Abkürzungsverzeichnis	57
Abbildungsverzeichnis	57
Tabellenverzeichnis	58

1 Einführung

Künstliche Intelligenz (KI) hat in den vergangenen Jahren einen rasanten Aufschwung erlebt und beeinflusst heute nahezu alle Branchen, von der Medizin über die Logistik bis hin zur Finanzwelt [A, S]. Auch die Softwareentwicklung befindet sich im Umbruch: Moderne KI-Verfahren eröffnen ein breites Spektrum neuer Einsatzfelder und verändern den Entwicklungsalltag grundlegend [Sie]. So unterstützt KI nicht mehr nur bei der Automatisierung von Routinetätigkeiten wie Codierung und Testing, sondern ermöglicht auch innovative Ansätze in der Fehlersuche, Qualitätssicherung und Projektsteuerung [A, Sie].

Mit diesen Potenzialen gehen jedoch weitreichende Fragestellungen und Herausforderungen einher. Neben technischen Aspekten wie Sicherheit, Robustheit und Code-Qualität rücken gesellschaftliche Dimensionen zunehmend in den Vordergrund, etwa die Frage nach ethischen Standards, Verantwortlichkeit und der Veränderung des Berufsbilds von Softwareentwickler:innen [Bra, Sch, Wei]. Insbesondere der Siegeszug generativer KI, beispielsweise durch leistungsstarke Large Language Models (LLMs), wirft komplexe Fragen zu Datenschutz, Fairness und der Integration in bestehende Entwicklungsprozesse auf [S, Wei].

Vor diesem Hintergrund nimmt die vorliegende Arbeit eine umfassende Perspektive ein: Ziel ist es, den Wandel der Softwareentwicklung unter dem Einfluss generativer KI systematisch zu beleuchten und zentrale Chancen wie Herausforderungen herauszuarbeiten. Im Mittelpunkt stehen dabei sowohl die Auswirkungen auf die konkrete Arbeitssituation von Entwickler:innen als auch die strategischen Implikationen für Unternehmen und Gesellschaft. Durch die Verbindung von Literaturrecherche, Praxisbeispiel und kritischer Reflexion sollen praxisrelevante Erkenntnisse sowie fundierte Handlungsempfehlungen abgeleitet werden.

1.1 Motivation

Die hohe Relevanz des Themas ergibt sich aus aktuellen Entwicklungen in Forschung und Praxis: Immer mehr Unternehmen setzen auf KI-Technologien, um Effizienzpotenziale und Innovationsschübe zu realisieren. Während generative KI, etwa durch automatisierte

Code-Generierung oder intelligente Projektsteuerung enorme Fortschritte und Produktivitätsgewinne verspricht, zeigen empirische Studien zugleich ein Spannungsverhältnis zwischen den erhofften Vorteilen und realen Risiken wie Intransparenz, Sicherheitslücken und ethischen Verzerrungen.

Nach aktuellen Schätzungen (Stand: 2025) erreicht der Softwaremarkt in Deutschland ein Volumen von rund 31 Milliarden US-Dollar. Entwickler:innen verbringen im Durchschnitt bis zu 17 Stunden pro Woche mit Wartungs- und Routineaufgaben, ein Indikator dafür, wie groß der Bedarf an Automatisierung und Qualitätsverbesserung in der Praxis ist. Genau hier setzt die generative KI an: Tools wie GitHub Copilot können durch automatisierte Boilerplate-Code-Generierung oder intelligente Code-Vervollständigung nicht nur die Produktivität erhöhen, sondern auch das Fachkräftethema teilweise entschärfen. Laut einer von Deloitte zitierten Studie lässt sich durch KI-basierte Coding-Tools die für Routineaufgaben benötigte Entwicklerzeit um bis zu 50 % reduzieren (vgl. [S] [Sie]).

Ein Beleg für die zunehmende wirtschaftliche Bedeutung ist der stetige Anstieg der Investitionen in KI-Technologien für die Softwareentwicklung in den letzten Jahren. Abbildung 1.1 veranschaulicht diese Entwicklung und unterstreicht, wie stark Unternehmen auf innovative KI-Lösungen setzen, um sich Wettbewerbsvorteile zu sichern.

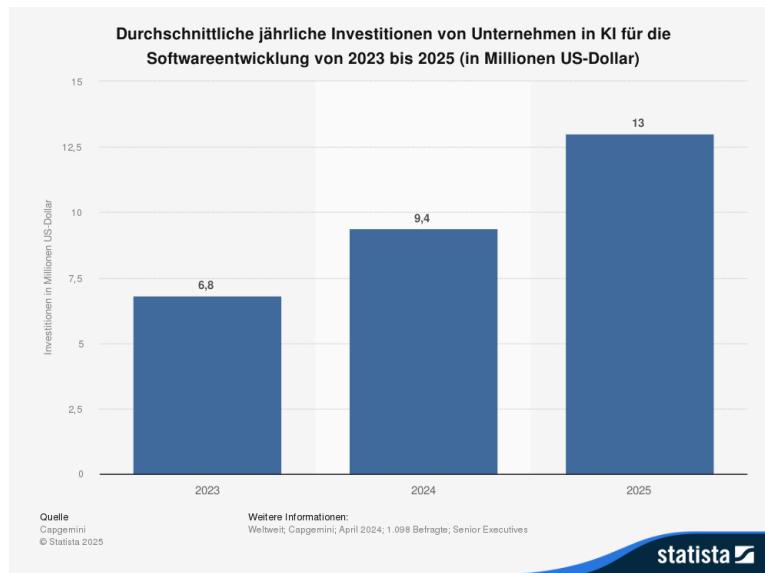


Abbildung 1.1: Jährliche Investitionen von Unternehmen in KI für Softwareentwicklung 2023–2025. Quelle: Capgemini [Cap24a].

Allerdings werfen diese Entwicklungen auch kontroverse Fragen auf: In welchem Maße verändert die zunehmende Abhängigkeit von KI-Tools die Rollen und Kompetenzen von Softwareentwickler:innen? Wie kann sichergestellt werden, dass durch KI-gestützte Automatisierung weiterhin qualitativ hochwertige, wartbare und sichere Software entsteht [Sie]? Hinzu kommt, dass die Softwareentwicklung durch agile Methoden wie Scrum

oder Kanban bereits stark dynamisiert ist. Die zusätzliche Integration von KI als Tool oder „Co-Entwickler“ erhöht die Anforderungen an Prozessgestaltung, Rollenverteilung und Qualitätsmanagement weiter.

1.2 Zielsetzung und Fragestellungen

Ziel dieser Arbeit ist es, die Auswirkungen generativer KI auf die Softwareentwicklung umfassend zu analysieren und daraus praxisnahe Handlungsempfehlungen für Unternehmen und Entwickler:innen abzuleiten. Im Fokus steht insbesondere, wie KI-gestützte Tools bestehende Entwicklungspraktiken verändern, welche Herausforderungen damit verbunden sind und wie sich Chancen und Risiken in der Praxis ausbalancieren lassen.

Daraus ergeben sich folgende zentrale Forschungsfragen:

- Wie verändert generative KI traditionelle Entwicklungspraktiken in der Softwareentwicklung?
- Welche spezifischen Herausforderungen entstehen durch KI-gestützte Softwareentwicklung hinsichtlich Sicherheit, Ethik und Code-Qualität?
- Wie kann generative KI Softwareentwickler:innen in einem agilen Entwicklungsprozess unterstützen?
- Wie lassen sich bestehende generative KI-Tools (wie Cursor, GitHub Copilot, v0) in den Entwicklungsprozess einer React-Native-App integrieren und welchen Einfluss hat dies auf Entwicklungszeit und Code-Qualität?

Um diese Fragen nicht nur theoretisch, sondern auch praxisnah zu beleuchten, wird im praktischen Teil der Arbeit eine React-Native-App als Fallstudie herangezogen. Ziel ist es, exemplarisch zu untersuchen, wie generative KI-Tools in reale Entwicklungsprozesse integriert werden können und welchen konkreten Mehrwert sie im Entwicklungsalltag bieten.

1.3 Methodik

Die Arbeit folgt einem literaturbasierten, qualitativen Ansatz, um ein umfassendes Bild der Potenziale und Herausforderungen generativer KI in der Softwareentwicklung zu zeichnen. Ziel ist es, die aktuelle Forschungslage systematisch zu erfassen, zu bewerten und praxisrelevante Schlüsse zu ziehen. Die Methodik gliedert sich in folgende Schritte:

1. **Literaturrecherche:** Umfassende Analyse wissenschaftlicher Publikationen aus IEEE Xplore, arXiv, SpringerLink sowie den in der Literaturdatenbank dieser Arbeit aufgeführten Fachquellen mit Fokus auf aktuelle Entwicklungen in der KI-gestützten Softwareentwicklung.
2. **Kategorisierung der Forschungsthemen:** Strukturierte Erfassung und Gruppierung zentraler Themenfelder wie Automatisierung, Produktivität, Sicherheit und ethische Fragestellungen.
3. **Vergleichende Analyse:** Systematischer Vergleich und Gegenüberstellung der Chancen und Herausforderungen auf Basis ausgewählter Studien und Fachbeiträge.
4. **Synthese und Ableitung von Schlussfolgerungen:** Entwicklung praxisorientierter Empfehlungen für Unternehmen und Entwickler:innen zur Integration von KI in der Softwareentwicklung.
5. **Praktische Demonstration:** Ergänzend zur Literaturauswertung wird ein Map-Screen (interaktive Kartenansicht) in der React Native-App „Locals“ exemplarisch implementiert. Dabei kommen verschiedene generative KI-Tools (u.a. Cursor, v0, GitHub Copilot) zum Einsatz, um die Unterstützung bei Code-Generierung, Testing und Qualitätsverbesserung zu evaluieren. Die gewonnenen Erkenntnisse aus diesem praktischen Teil werden systematisch mit den theoretischen Ergebnissen abgeglichen.

Dieses methodische Vorgehen ermöglicht es, den aktuellen Forschungsstand kritisch einzurichten, praxisnahe Einblicke zu gewinnen und fundierte Empfehlungen für die erfolgreiche Nutzung generativer KI in der Softwareentwicklung abzuleiten.

1.4 Abgrenzung

Die Arbeit konzentriert sich auf die theoretische Analyse der Chancen und Herausforderungen von KI in der Softwareentwicklung. Folgende Aspekte werden bewusst ausgeklammert:

- **Technische Implementierungen:** Es werden keine eigenen KI-Modelle oder neuen Algorithmen entwickelt.
- **Empirische Studien:** Die Arbeit basiert auf einer literaturgestützten Analyse; eigene Befragungen, Experimente oder quantitative Erhebungen werden nicht durchgeführt.

- **Rechtliche Rahmenbedingungen:** Eine detaillierte Untersuchung rechtlicher oder regulatorischer Aspekte findet nicht statt.

Obwohl ein begrenzter praktischer Teil in Form einer exemplarischen Funktionsimplementierung integriert wird, dient dieser ausschließlich als Proof of Concept zur Veranschaulichung des KI-Einsatzes. Eine umfassende empirische oder technische Evaluation erfolgt nicht.

Diese Abgrenzungen und methodischen Einschränkungen sind bei der Interpretation der Ergebnisse sowie im abschließenden Fazit und Ausblick dieser Arbeit stets zu berücksichtigen.

2 Theoretische Grundlagen

Dieses Kapitel legt die zentralen theoretischen und technologischen Grundlagen der Arbeit. Im Mittelpunkt stehen die Definition generativer KI, die zugrundeliegenden Algorithmen und Modelle sowie ausgewählte Praxisbeispiele für den Einsatz KI-gestützter Tools in der Softwareentwicklung. Diese Grundlagen sind essenziell, um die Potenziale und Herausforderungen generativer KI im weiteren Verlauf der Arbeit differenziert bewerten zu können.

2.1 Einordnung generativer KI in die Softwareentwicklung

Künstliche Intelligenz (KI) leitet einen grundlegenden Wandel in der Softwareentwicklung ein. Die aktuelle, rechtlich verbindliche Definition der Europäischen Union beschreibt KI als maschinengestützte Systeme, die mit unterschiedlichem Grad an Autonomie Vorhersagen, Empfehlungen oder Entscheidungen generieren, welche physische oder virtuelle Umgebungen beeinflussen können [noa]. Diese Definition etabliert sich zunehmend als Referenzrahmen in Forschung und Praxis.

Moderne KI-Systeme unterscheiden sich deutlich von traditionellen softwarebasierten Ansätzen. Während frühe KI-Tools vor allem Aufgaben wie Syntaxprüfung unterstützen, übernehmen generative KI-Systeme (GenAI) heute komplexe Aufgaben im Design, der Entwicklung und Wartung von Software, etwa durch die automatische Generierung von Code-Snippets, Unterstützung bei Unit-Tests oder die Automatisierung von Deployment-Prozessen [Don].

Zu den wichtigsten Architekturen zählen Transformer-Modelle (insbesondere Large Language Models wie GPT), Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs) und Diffusion Models. Während LLMs primär für Text- und Codegenerierung eingesetzt werden, kommen GANs und Diffusion Models vor allem in der Bild- und Medienerzeugung zum Einsatz [Don]. Viele aktuelle Tools wie ChatGPT, GitHub Copilot oder Stable Diffusion basieren auf diesen Architekturen und treiben Innovationen in der Softwareentwicklung maßgeblich voran.

Der Einsatz generativer KI verändert Methoden und Arbeitsweisen grundlegend. Besonders Large Language Models prägen sämtliche Phasen des Softwareentwicklungszyklus, von der Anforderungsanalyse bis zur Umsetzung und Wartung. Entscheidungsunterstützung, die Rekonstruktion von Softwarearchitekturen sowie Methoden wie Few-Shot-Prompting und Retrieval-Augmented Generation (RAG) sind dabei zentrale Elemente [Esp].

Trotz fortschreitender Automatisierung bleibt die sorgfältige Validierung der von KI generierten Ergebnisse durch Entwickler:innen unerlässlich. Die größten Herausforderungen betreffen Präzision und Verlässlichkeit der Modelle, den Umgang mit sogenannten „Halluzinationen“, ethische Aspekte sowie das Fehlen domänenspezifischer Benchmarks und Standards [Esp].

Die Entwicklung der Softwaretechnik unter dem Einfluss von KI lässt sich grob in drei Phasen gliedern. Während Software Engineering 1.0 klassische, code-zentrierte Entwicklungspraktiken beschreibt, kennzeichnet SE 2.0 die Integration KI-gestützter Assistenzsysteme (z. B. Copilots) in etablierte Prozesse. Die Vision von SE 3.0 sieht einen Paradigmenwechsel zu einer intent-basierten, dialogorientierten Entwicklung vor, in der KI-Systeme als intelligente Partner agieren [Has].

	SE 1.0 (Vergangenheit)	SE 2.0 (Gegenwart)	SE 3.0 (Zukunft)
Leitmotiv	Code-first	Code-first + KI-Unterstützung	Intent-first, KI als Partner
Technologie	Klassische Tools, Programmalyse	KI-gestützte Copilots, Foundation Models	Konversationsorientiert, Reasoning-Modelle
Rolle von Mensch/KI	Mensch zentral, alles manuell	Mensch + KI, KI assistiert	Symbiose: Mensch & KI kollaborieren
Besonderheiten	Fokus auf Code, klare Abläufe	Effizienzsteigerung, hohe kognitive Last	Dialog, Automatisierung, wissengetrieben

Tabelle 2.1: Entwicklung der Softwaretechnik unter KI-Einfluss (SE 1.0 bis SE 3.0).
Quelle: Eigene Darstellung in Anlehnung an Hassan et al. [Has].

2.1.1 Beispiele für generative KI-Tools in der Praxis

Generative KI-Tools sind mittlerweile aus der Softwareentwicklung nicht mehr wegzudenken und decken ein breites Spektrum an Aufgaben ab – von der Code-Vervollständigung über automatische Testgenerierung bis hin zur Unterstützung ganzer Entwicklungsprojekte [Don]. Zu den praxisrelevanten Vertretern zählen unter anderem **GitHub Copilot**, **TabNine**, **Cursor AI** und **Devin AI**.

GitHub Copilot ist ein KI-basierter Codeassistent, der Entwickelnden direkt im Kontext der Entwicklungsumgebung Vorschläge für Code-Snippets, komplette Funktionen und

sogar Tests unterbreitet. Die Integration erfolgt nahtlos in gängige IDEs wie Visual Studio Code, IntelliJ oder Eclipse. Das zugrunde liegende Modell – OpenAI Codex – wird mit Kommentaren oder natürlicher Sprache angesteuert. Typische Anwendungsfelder sind schnelles Prototyping, die Generierung von Boilerplate-Code und die Unterstützung beim Onboarding neuer Teammitglieder [Don]. Praxiserfahrungen zeigen, dass Copilot die Entwicklung – etwa von React-Anwendungen – erheblich beschleunigen kann, indem es gezielte Codevorschläge für Authentifizierung, Routing oder Formularvalidierung liefert und bei der Fehlerbehebung unterstützt. Dennoch bleibt eine kritische Überprüfung der KI-Vorschläge unerlässlich [Ker].

TabNine ist ein weiteres KI-gestütztes Tool zur Code-Vervollständigung, das ursprünglich auf GPT-2 basierte und mittlerweile ein eigenes Modell verwendet. Es generiert Codevorschläge in Echtzeit für zahlreiche Programmiersprachen und passt sich sukzessive dem Stil der jeweiligen Entwicklerperson an. TabNine unterstützt alle gängigen Entwicklungsumgebungen und bietet zusätzlich eine Chat-Funktion für gezielte Code-Fragen. Die Flexibilität durch wahlweise lokale oder cloudbasierte Modelle wird besonders im Hinblick auf unterschiedliche Datenschutzanforderungen geschätzt [Don].

Cursor AI steht für die nächste Generation KI-basierter Entwicklungstools. Es kann auf Grundlage natürlicher Sprache vollständige Applikationen generieren und nutzt dabei fortschrittliche Ansätze wie Retrieval-Augmented Generation (RAG) und Agentic AI. Die Stärke von Cursor AI liegt in der End-to-End-Generierung kompletter Projekte, was insbesondere für schnelles Prototyping oder den Aufbau komplexer Softwarelösungen von Vorteil ist [Don].

Devin AI geht noch einen Schritt weiter und versteht sich als „AI Software Engineer“. Das Tool setzt komplett Softwareprojekte auf Basis natürlicher Sprache um, bricht Anforderungen in Aufgaben herunter, automatisiert Testprozesse und erstellt Deployment-Skripte. Besonders hervorzuheben ist die Fähigkeit von Devin, langfristige Planungen umzusetzen und kontinuierliche Anpassungen an neue Anforderungen vorzunehmen [Don].

Typische Einsatzszenarien

In der Praxis kommen diese Tools in verschiedenen Bereichen zum Einsatz:

- **Code-Generierung und Vervollständigung:** Automatisiertes Schreiben von Code, Vorschläge für Funktionen, Klassen oder API-Integrationen.

- **Test- und Debugging-Unterstützung:** Generierung von Unit- und Integrations-Tests, Erkennung von Fehlern und Vorschläge für Bugfixes.
- **Projekt-Scaffolding und Boilerplate:** Automatisches Erstellen von Grundstrukturen für neue Projekte.
- **End-to-End-Entwicklung:** Vollständige Umsetzung von Projektanforderungen inklusive Deployment-Skripten und CI/CD-Konfiguration [Don].

Vorteile und Grenzen in der Praxis

Die Integration generativer KI-Tools führt nachweislich zu erheblichen Zeitersparnissen, konsistenterem Code und einer schnelleren Einarbeitung neuer Teammitglieder. Gleichzeitig bleiben strukturierte Review-Prozesse und ein kritischer Umgang mit KI-generierten Vorschlägen unverzichtbar, um Qualitäts- und Sicherheitsrisiken zu minimieren. Fortgeschrittene Werkzeuge wie Cursor AI oder Devin AI bieten ein hohes Maß an Automatisierung, sind jedoch häufig kostenintensiv und nicht in jedem Anwendungsfall ausgereift [Don].

2.1.2 Wichtige Algorithmen und Modelle in der Softwareentwicklung

Aktuelle Forschung betont, dass ein zukunftsorientiertes Software-Ökosystem nicht nur technologische Innovation, sondern auch eine optimierte Zusammenarbeit von Mensch und KI erfordert. Ein systematischer Umgang mit technischer Verschuldung sowie die gezielte Nutzung externer Wissensquellen gelten als Schlüsselfaktoren moderner Entwicklungsframeworks [Mat]. Darüber hinaus zeigen empirische Studien, dass KI-Assistenzsysteme die Codequalität und Wartbarkeit nachweislich verbessern können [Marb]. Der Ansatz des sogenannten „AI-Native Software Engineering“ (SE 3.0) fokussiert eine enge Verzahnung von KI, Entwicklerkompetenz und Geschäftsprozessen und steht für einen Wandel hin zu einer kooperativen, dialogorientierten Softwareentwicklung [Has].

Im Kern basieren moderne KI-Tools wie GitHub Copilot, Cursor oder v0 auf sogenannten Large Language Models (LLMs) und verwandten Architekturen wie Transformers. Diese tiefenlernenden neuronalen Netze werden auf großen Mengen an Quellcode und natürlicher Sprache trainiert und nutzen Mechanismen wie Self-Attention, um den Kontext über längere Sequenzen hinweg zu erfassen. So gelingt es den Modellen, sowohl syntaktisch als auch semantisch komplexe Strukturen zu erkennen und zu generieren, etwa im Bereich der Code-Logik oder bei der Anwendung typischer Designmuster [ND, Esp]. Während Diffusionsmodelle in der Bild- und Mediengenerierung dominieren,

bleiben für text- und codebasierte Softwareentwicklung weiterhin LLMs und Transformer-Architekturen zentral [Wei].

In der praktischen Anwendung nutzen moderne Assistenzsysteme typischerweise spezialisierte Sprachmodelle wie OpenAI Codex, Code Llama oder StarCoder, die gezielt auf Programmcode vorgenutzt wurden. Diese Modelle sind in der Lage, ausgehend vom Kontext, etwa bestehender Code, Kommentare oder Projekthistorie, automatisiert neue Code-Abschnitte zu generieren, Fehlerkorrekturen vorzuschlagen oder passende Testfälle zu erstellen [Cou, Esp]. Ihr Funktionsspektrum reicht von der automatischen Erstellung kompletter Funktionen und Module auf Basis kurzer Beschreibungen in natürlicher Sprache über die Generierung von Unit-Tests und die Unterstützung beim Review bis hin zur Entwicklung und Auswahl geeigneter Softwarearchitekturen oder Design Patterns, insbesondere im projektspezifischen Kontext [Cou, Esp, Don].

LLMs übernehmen damit zunehmend Aufgaben, die von der reinen Code-Generierung bis hin zu komplexeren Anforderungen reichen: Sie erstellen automatisiert Tests, variieren Testdaten, erkennen typische Fehlerbilder und unterstützen Entwickler:innen durch Vorschläge zur Architektur oder durch das Übersetzen von Requirements in konkrete Design-Entwürfe oder Diagramme [Esp, ND].

Trotz des enormen Potenzials solcher Modelle sind wesentliche Herausforderungen zu beachten. So können LLMs zwar häufig syntaktisch korrekten Code generieren, dieser ist jedoch nicht immer inhaltlich passend oder sicher, insbesondere bei vagen Prompts oder fehlendem Kontext (*Halluzinationen*). Die Erklärbarkeit und Transparenz der generierten Vorschläge bleibt oftmals eingeschränkt, da die Entscheidungswege der Modelle schwer nachvollziehbar sind. Ohne gezieltes Fine-Tuning auf spezifische Projekte oder Domänen bleibt das Wissen zudem meist allgemein und kann individuellen Anforderungen nicht immer gerecht werden [Esp, ND, Don].

Im Praxisteil dieser Arbeit werden die beschriebenen Modelle durch Tools wie GitHub Copilot, Cursor und Bolt eingesetzt, um Entwickler:innen in sämtlichen Phasen der Softwareentwicklung, von Architekturentwurf bis Testing, aktiv zu unterstützen. Diese Werkzeuge etablieren sich damit zunehmend als kollaborative Partner und verändern klassische Entwicklungsprozesse nachhaltig [Esp, ND].

2.2 Generative KI-Tools: Funktion und Anwendung

Generative KI-Tools wie GitHub Copilot, Cursor oder v0 prägen den modernen Softwareentwicklungsprozess entscheidend. Ihre Hauptfunktion besteht darin, natürliche Sprache, sogenannte Prompts, in ausführbaren Code, Testfälle oder Dokumentationen umzusetzen. Damit verändern sie sowohl technische Workflows als auch die Zusammen-

arbeit in Entwicklungsteams und stellen neue Anforderungen an die Kompetenzen der Beteiligten [Wei].

Die zugrundeliegenden Large Language Models (LLMs) ermöglichen ein Prompt-basiertes Entwicklungsparadigma: Entwickler:innen beschreiben Aufgaben in natürlicher Sprache und erhalten daraufhin passende Vorschläge. Diese erscheinen entweder als *Code Completion* direkt beim Tippen oder als vollständige Funktionsblöcke [Ker, Wei]. Neuere Ansätze wie SENAI integrieren generative KI von Beginn an in den Software-Engineering-Prozess und erlauben so eine hochautomatisierte, KI-zentrierte Entwicklung [Saa].

Systematische Literaturübersichten zeigen, dass die Integration generativer KI in Entwicklungsumgebungen das Nutzererlebnis, die Akzeptanz und den praktischen Nutzen maßgeblich beeinflusst. Besonders Faktoren wie Transparenz, Usability und intelligente Feedbackmechanismen sind entscheidend für den Erfolg im Alltag [Ser]. Die praktische Nutzung generativer KI erfolgt heute meist direkt über Plugins für etablierte IDEs wie Visual Studio Code oder JetBrains, oftmals auch über API-Schnittstellen. Dadurch lassen sich Aufgaben wie Refactoring, Testing oder Dokumentation unmittelbar und nahtlos in die gewohnte Arbeitsumgebung integrieren [Ker, Shi, Wei]. Besonders hervorzuheben ist zudem, dass KI-gestützte Assistenzsysteme neue Teilhabemöglichkeiten für sehbeeinträchtigte Entwickler:innen eröffnen können, vorausgesetzt, die Tools sind barrierefrei gestaltet [FS].

Ein typischer Workflow zeigt sich etwa beim Pair Programming mit Copilot: Aufgaben werden in Form von Prompts gestellt, das Tool generiert passende Codevorschläge, die geprüft, angepasst oder verworfen werden können. Studien belegen, dass diese Arbeitsweise Routineaufgaben wie Testautomatisierung, Refactoring und Dokumentation signifikant beschleunigt [Ker, Wei, Shi]. In agilen Teams kann generative KI zudem einen Beitrag zur Qualitätsbewertung und zur Dokumentation von Anforderungen leisten [Gey].

Der Einsatz generativer KI bietet zahlreiche Vorteile, die in aktuellen Studien hervorgehoben werden: So wird insbesondere die Automatisierung von Routineaufgaben und die damit verbundene Zeitersparnis hervorgehoben, genauso wie die Verbesserung der Codequalität durch die Erkennung häufiger Fehler und gezielte Empfehlungen für Best Practices. Außerdem werden niedrigere Einstieghürden für weniger erfahrene Entwickler:innen genannt, denen durch kontextbasierte Vorschläge der Zugang zur Entwicklung erleichtert wird [Don, Ser]. Darüber hinaus fördern KI-gestützte Code Reviews die Kollaboration zwischen Mensch und Maschine und erhöhen die Transparenz im Entwicklungsprozess [Ala]. Durch kontinuierliche Modellverbesserung und flexible Integration in unterschiedliche Projekte kann das Optimierungspotenzial weiter gesteigert werden [Ker, Wei].

Trotz dieser Vorteile bestehen zentrale Herausforderungen. Ein häufig diskutiertes Problem sind sogenannte *Halluzinationen*: Modelle generieren zwar syntaktisch korrekten, inhaltlich aber fehlerhaften oder unsicheren Code, besonders bei unscharfen oder vagen Prompts [Shi, Wei]. Hinzu kommen Bias und Kontextdefizite, also die Übernahme von Vorurteilen aus den Trainingsdaten oder das Nichtbeachten projektspezifischer Regeln. Ein weiteres Risiko besteht in der Generierung unsicheren Codes, etwa durch Vorschläge mit hardcodierten Zugangsdaten, die explizit überprüft werden müssen. Zudem beobachten Studien ein übermäßiges Vertrauen vieler Entwickler:innen in die Vorschläge der KI, eine unkritische Übernahme ohne manuelle Überprüfung kann zu schwerwiegenden Fehlern führen [Shi, Wei].

Um einen produktiven und sicheren Einsatz zu gewährleisten, empfiehlt die Literatur spezifische Designprinzipien für generative KI-Tools [Wei]: Erstens sollte die Gestaltung der Systeme so erfolgen, dass Nutzer:innen die Funktionsweise und Grenzen nachvollziehen können (*Design for Mental Models*). Zweitens müssen Feedbackmechanismen sowohl Vertrauen fördern als auch zur kritischen Prüfung anregen (*Design for Appropriate Trust & Reliance*). Drittens ist die Berücksichtigung von Fehlern essenziell: Tools sollten aktiv auf mögliche Fehler hinweisen und die Nutzer:innen zur Korrektur befähigen (*Design for Imperfection*). Die Gestaltung generativer GUIs und Entwicklungsprozesse muss diese Prinzipien berücksichtigen, um eine nachhaltige und verantwortungsvolle Integration zu ermöglichen [Lee, Che, Gil]. Flexibilität, Feedback und kontinuierliche Anpassung sind hier Schlüsselfaktoren.

Insgesamt zeigt sich, dass der Mehrwert generativer KI-Tools vor allem dann zum Tragen kommt, wenn sie sinnvoll in bestehende Entwicklungsumgebungen integriert, kritisch überprüft und auf die jeweiligen Team- und Projektanforderungen angepasst werden.

3 Praktische Demonstration

Die bisherigen Kapitel haben die grundlegenden Technologien, Chancen und Herausforderungen generativer KI in der Softwareentwicklung theoretisch beleuchtet. Im folgenden Kapitel wird dieses Wissen nun praktisch angewendet: Anhand der exemplarischen Implementierung einer interaktiven Kartenansicht („Map-Screen“) in der React Native-App *Locals* wird demonstriert, wie verschiedene generative KI-Tools (z. B. GitHub Copilot, Cursor, Bolt) den Entwicklungsprozess konkret unterstützen können. Ziel ist es, die praktische Integration, Arbeitsweise sowie Vor- und Nachteile der Tools unter realistischen Bedingungen zu untersuchen. Das Kapitel gliedert sich in die Vorstellung des Projekts, die Umsetzung mit den KI-Tools und eine vergleichende Bewertung der Ergebnisse.

3.1 Zielsetzung und Vorgehen

Das Ziel dieses Kapitels ist es, die Potenziale und Herausforderungen generativer KI-Tools in der Softwareentwicklung anhand eines praxisnahen Beispiels systematisch zu evaluieren. Am Beispiel der exemplarischen Implementierung einer interaktiven Kartenansicht („Map-Screen“) in der mobilen App „Locals“ wird untersucht, wie moderne KI-gestützte Tools den Entwicklungsprozess unterstützen, in welchen Bereichen sie ihre Stärken ausspielen können und wo ihre Grenzen sichtbar werden. Aufbauend auf den zuvor beschriebenen theoretischen Grundlagen bietet dieses Praxisbeispiel die Gelegenheit, zentrale Konzepte und Annahmen empirisch zu überprüfen und kritisch zu reflektieren.

Die Entscheidung, den Fokus auf die Entwicklung einer interaktiven Kartenansicht zu legen, ist aus mehreren Gründen besonders sinnvoll. Erstens handelt es sich bei Map-Views um zentrale und technisch anspruchsvolle Features in Event- und Social-Apps, da sie die Integration verschiedener Technologien wie Geolocation, Datenmanagement, UI/UX-Design und Filterfunktionen erfordern. Zweitens vereint diese Aufgabe klassische Frontend-Herausforderungen, beispielsweise im Bereich State-Management und UI-Logik, mit der Integration externer Libraries wie `react-native-maps` oder der Google Maps API. Gerade durch diese Vielschichtigkeit eignet sich die Map-View besonders gut, um die Leistungsfähigkeit generativer KI-Tools im realen Entwicklungsprozess

kritisch zu beleuchten. Nicht zuletzt unterstreicht auch der unmittelbare Nutzen für die Anwender:innen die hohe Praxisrelevanz des gewählten Beispiels.

Für die Umsetzung wurden drei führende KI-basierte Entwicklungstools ausgewählt, die aktuelle Trends und Methoden im Bereich der KI-gestützten Softwareentwicklung abbilden. Zum einen kommt **GitHub Copilot** zum Einsatz, ein kontextsensitiver Code-Assistent mit Echtzeit-Vervollständigung, der automatische Vorschläge für Funktionen, Tests und Dokumentation generiert und sich direkt in gängige IDEs integriert [Git]. Ergänzend wird **Cursor** verwendet, ein KI-basierter Editor, der durch Multi-Line-Edits, Prompt Chaining, Fehlerdiagnose und eine dialogorientierte Benutzeroberfläche die Umsetzung komplexer Aufgaben unterstützt und plattformübergreifend einsetzbar ist [Cur]. Als drittes Tool kommt **Bolt.new** hinzu, eine cloudbasierte Entwicklungsumgebung, die ohne lokale Installation auskommt und die Entwicklung sowie das Deployment direkt aus Prompts heraus ermöglicht. Bolt.new unterstützt mehrere Plattformen, bietet Live-Bearbeitung und Teamfunktionen [Bol].

Die vergleichende Analyse dieser drei Tools ermöglicht es, unterschiedliche Formen der Interaktion zwischen Mensch und KI im realen Entwicklungsalltag zu bewerten, von klassischen Code-Vervollständigern wie Copilot über dialogorientierte Systeme wie Cursor bis hin zu agentenbasierten Komplettumgebungen wie Bolt.new. Dadurch lassen sich Stärken, Schwächen und bestmögliche Einsatzszenarien generativer KI-Tools in der Praxis gezielt herausarbeiten.

3.2 Vorstellung der App „Locals“

Die App *Locals* ist als mobile Plattform konzipiert, die es Nutzer:innen ermöglicht, lokale Events zu entdecken, zu erstellen und zu verwalten. Zielgruppe sind vor allem junge, urbane Menschen, die soziale Interaktionen rund um Veranstaltungen suchen. Die technische Umsetzung setzt auf eine moderne, modulare und plattformübergreifende Architektur, die sowohl schnelle Entwicklungszyklen als auch eine hohe Flexibilität bei der Erweiterung von Funktionen erlaubt.

Im Frontend kommen React Native und TypeScript zum Einsatz, ergänzt durch das Expo-Framework, das eine konsistente Entwicklung für iOS- und Android-Geräte ermöglicht. Für die Backend-Funktionalitäten wie Authentifizierung, Datenhaltung und Synchronisation wird Firebase genutzt. Die Navigation innerhalb der App basiert auf den Bibliotheken `@react-navigation/native` und `expo-router`. Das State-Management wird über eigene Context-Provider wie `AuthProvider` und `EventsProvider` realisiert. Für die Benutzeroberfläche werden Icon-Bibliotheken wie `@expo/vector-icons` und

`lucide-react-native` verwendet. Karten- und Standortfunktionen werden durch `react-native-maps` und `expo-location` integriert.

Die Anwendung gliedert sich in drei Hauptbereiche: Der Explore-Screen dient als Event-Feed, der die Veranstaltungen nach Standort und Interessen filtert. Der Map-Screen stellt eine interaktive Kartenansicht bereit, in der Events als Marker angezeigt und mit Hilfe von Filtern durchsucht werden können. Im Profil-Screen erhalten die Nutzer:innen eine Übersicht und Verwaltung ihrer eigenen sowie besuchten Events und Profileinstellungen. Im Rahmen dieser Arbeit wird insbesondere der Map-Screen als prototypisches, KI-gestütztes Feature exemplarisch entwickelt und dient so als Grundlage für die weitere Analyse der Potenziale generativer KI in der Softwareentwicklung.

Eine zentrale Rolle in der Architektur der App übernimmt das `RootLayout`, das sowohl die benötigten Schriftarten als auch Authentifizierungs- und Event-Kontexte lädt. Die Steuerung der Nutzerführung ist strikt zustandsbasiert und erfolgt über die Hooks `useAuth`, `useSegments` und `useRouter`. Dieses Architekturprinzip gewährleistet eine klare Trennung zwischen Zustandsverwaltung und Routing und erleichtert so spätere Erweiterungen oder Integrationen, beispielsweise von KI-gestützten Funktionen.

Bereits vor der Integration generativer KI-Tools wurden zentrale Funktionen implementiert. Dazu zählen die Benutzerauthentifizierung mit Firebase Authentication, eine umfassende Profilverwaltung für eigene und besuchte Events sowie eine vollständige Eventverwaltung mit Funktionen zum Anlegen, Bearbeiten und Löschen von Veranstaltungen. Die Navigation innerhalb der App erfolgt über ein Tab-System, das einen schnellen Wechsel zwischen den Hauptbereichen Explore, Map und Profil ermöglicht. Ein responsives Design sorgt zudem für eine einheitliche Darstellung auf sämtlichen Endgeräten.

Die modulare Struktur und die klare Abgrenzung der einzelnen App-Komponenten schaffen eine ideale Grundlage, um den gezielten Einsatz generativer KI-Tools im Entwicklungsprozess zu evaluieren und deren Auswirkungen auf Wartbarkeit, Erweiterbarkeit und Benutzerfreundlichkeit zu untersuchen.

3.3 Implementierung der interaktiven Kartenansicht mit KI-Unterstützung

3.3.1 Prompt-Setup: Einheitliche Aufgabenstellung für alle Tools

Um eine objektive und vergleichbare Bewertung zu ermöglichen, wurde für die Implementierung des Map-Screens in allen drei Tools (*GitHub Copilot*, *Cursor*, *Bolt.new*) derselbe,

detaillierte Prompt verwendet, der die funktionalen und technischen Anforderungen klar definierte:

```
1 Create a Map Screen with Event Markers and Filter in React Native
2
3 Create a React Native component called `MapScreen` that displays
   event markers on a map using event data from the `EventsProvider`
   context.
4
5 ## Requirements
6
7 ### Functionality
8 - Use the list of events from the `EventsProvider` context.
9   (Access: `const { events, loading, error, refreshEvents } =
   useEvents();`)
10 - Each event contains:
11   - `docId: string`
12   - `title: string`
13   - `category: string`
14   - `date: string`
15   - `geoPoint: { latitude: number; longitude: number }`
16 - Example Context Usage: const { events, loading, error,
   refreshEvents } = useEvents();
17 - Display all events as markers on a map (`react-native-maps` or Expo
   MapView).
18 - When a marker is tapped, show a callout or modal with event details
   (`title`, `time`, `category`, optional image/avatar).
19 - Add filter options above the map (e.g. by category, date, distance)
20   .
21   Only display events that match active filters.
22 - Handle loading and error states appropriately.
23
24 ### UI/UX
25 - Modern mobile UI with a clean filter bar on top, responsive marker
   popups, and a loading spinner.
26 - Use functional components and React hooks (TypeScript).
27 - Styling should be clean and minimal (simple css with StyleSheet
   preferred).
28 - Map should fit all visible markers and allow user zoom/pan.
29 - Compatible with Expo.
30
31 ### File Structure
32 - Main screen in `map.tsx`.
33 - Optional: Reusable `EventMarker` component.
34
35 ### Additional
36 - Keep code modular, clean, and well-commented.
37 - Focus on maintainability and clarity.
38 - Use the attached layout (filter bar on top, map below, details as
   bottom sheet/callout) as a visual reference.
```

```
38
39     /** Example Event Type (TypeScript)
40     interface Event {
41         docId: string;
42         title: string;
43         category: string;
44         date: string;
45         geoPoint: { latitude: number; longitude: number };
46         // ...more fields possible
47     }
48
49     ## Goal
50     A fully functional, modern React Native map screen with event markers
      , filter bar, marker popups, and proper handling of loading/error
      states -- ready to integrate into an Expo app.
```

Hinweis: Diese Aufgabenstellung wurde in allen Demonstrationen identisch genutzt, um die Unterschiede in Lösungsstrategie und Codequalität der Tools direkt vergleichbar zu machen.

3.3.2 Demonstration mit GitHub Copilot

Setup und Vorgehen

Die Entwicklung des Map-Screens wurde exemplarisch mit **GitHub Copilot** in Visual Studio Code durchgeführt. Für größtmögliche Vergleichbarkeit kamen ausschließlich die Copilot-Funktionen zum Einsatz, keine weiteren KI-Plugins.

Zu Beginn wurde die Entwicklungsumgebung vorbereitet (z. B. Installation von `react-native-maps` und `expo-location`). Die Aufgabenstellung wurde als Kommentar oder Docstring auf Englisch eingefügt (*siehe Abschnitt 3.3.1*).

Schrittweise Umsetzung und Reflexion

Die Entwicklung erfolgte nach folgendem Muster:

1. **Prompt definieren:** Pro Feature (z. B. Marker, Filter, Event-Details) wurde ein spezifischer Kommentar als Arbeitsanweisung eingefügt.
2. **Vorschläge von Copilot akzeptieren oder anpassen:** Vorschläge wurden übernommen, angepasst oder verworfen.

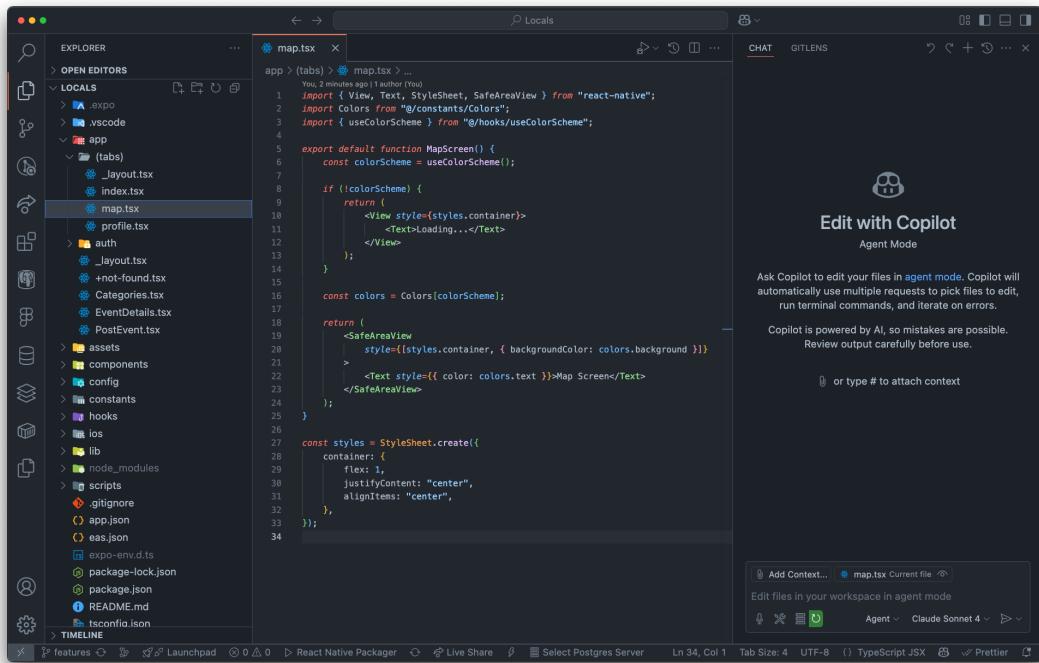


Abbildung 3.1: Ausgangszustand der Anwendung vor dem Einsatz von Copilot.
Copilot-Demo

3. **Test und Dokumentation:** Nach jeder Änderung wurde der Code getestet und die Funktionsweise reflektiert.
4. **Fehlersuche und Nacharbeit:** Fehlerhafte Vorschläge oder Bugs wurden durch Rücksprache mit Copilot, Recherche oder manuelle Nacharbeit behoben.

Die technische Umsetzung umfasste:

- Anzeige aller Events als Marker (mit Kategorie und Titel) auf der Karte.
- Filterleiste zur Auswahl nach Kategorie und Datum.
- Darstellung von Event-Details beim Tippen auf einen Marker.
- Responsives Layout, modernes UI-Design, Fehler- und Ladezustände.

Die Code-Generierung mit Copilot erfolgte modular und war in den meisten Fällen nachvollziehbar. Das Tool erstellte automatisch das Grundgerüst der Map-Komponente und ergänzte Schritt für Schritt die notwendige Logik für Marker, Filter und Event-Details. Bei komplexeren Aufgaben oder weniger klar formulierten Prompts zeigte sich jedoch, dass die Vorschläge von Copilot nicht immer transparent oder unmittelbar erklärbar waren. Einzelne Schwächen und typische Fehlerquellen, die im weiteren Verlauf noch detaillierter ausgeführt werden, wurden im praktischen Einsatz sichtbar.

I've implemented the Map Screen with Event Markers and Filtering functionality as requested. Let me explain the key features:

Key Components Implemented

1. Map with Event Markers
 - Uses `react-native-maps` to display a Google Maps interface
 - Events from the `EventsProvider` context are displayed as interactive markers
 - Each marker shows the first letter of the event category with a custom styling
 - Markers include callouts that show basic event information when tapped
2. Filter Functionality
 - Two types of filters:
 - **Category Filter:** Allows filtering events by categories like Music, Sports, Food, etc.
 - **Date Filter:** Allows filtering events by timeframe (Today, This Week, This Month)
 - Toggle buttons to switch between filter types
 - Horizontal scrollable filter chips for easy selection

3. Event Details Modal

- Detailed view of events when a marker callout is tapped
- Shows event title, image (if available), category, date, location, and description
- Clean, responsive design with proper handling of different screen sizes

4. Loading & Error States

- Displays loading indicator while events are being fetched
- Shows appropriate error message with a retry button if events fail to load

5. Additional Features

- Map automatically fits to display all visible markers
- Event counter shows the number of filtered events currently displayed
- Support for both light and dark modes using your existing Colors and theme system
- Responsive design that works across different screen sizes

Abbildung 3.2: Rückmeldung und Hinweise von Copilot während der Implementierung.
Copilot-Demo

Zu den wesentlichen Stärken von Copilot zählte insbesondere die effiziente Generierung von Boilerplate-Code und wiederkehrenden Patterns, wodurch der Entwicklungsprozess spürbar beschleunigt wurde. Das Tool ermöglichte schnelle und meist treffende Vorschläge für UI-Komponenten sowie für die grundlegende Interaktionslogik der App. Auch in Bezug auf die Erkennung einfacher Fehler und die automatische Anpassung von Typen zeigte sich Copilot im Praxistest als zuverlässig und hilfreich.

Gleichzeitig wurden auch verschiedene Schwächen und typische Fehlerquellen deutlich. So generierte Copilot gelegentlich fehlerhafte oder veraltete Import-Pfade, was sich vor allem bei weniger verbreiteten Packages oder nach Updates im Projekt bemerkbar machte. Darüber hinaus kam es vereinzelt zu Missverständnissen bei nicht exakt spezifizierten Datenstrukturen, was dazu führte, dass bestimmte Vorschläge nicht wie erwartet funktionierten. Die Filter-Logik, insbesondere für den „All“-Filter, bereitete zu Beginn Schwierigkeiten und musste manuell nachgebessert werden. Insgesamt zeigte sich, dass bei komplexeren Anforderungen fast immer eine zusätzliche Nacharbeit notwendig war, um die gewünschten Resultate zu erzielen und eine stabile App-Funktionalität sicherzustellen.

Im Rückblick erwiesen sich die Vorschläge von Copilot bei Standardaufgaben als überwiegend brauchbar, wobei die subjektive Zufriedenheit bei etwa 4 von 5 Punkten lag. Bei anspruchsvoller Anforderungen, etwa beim State-Management oder bei spezifischen Typ-Logiken, blieben die generierten Vorschläge jedoch häufig unvollständig. Die

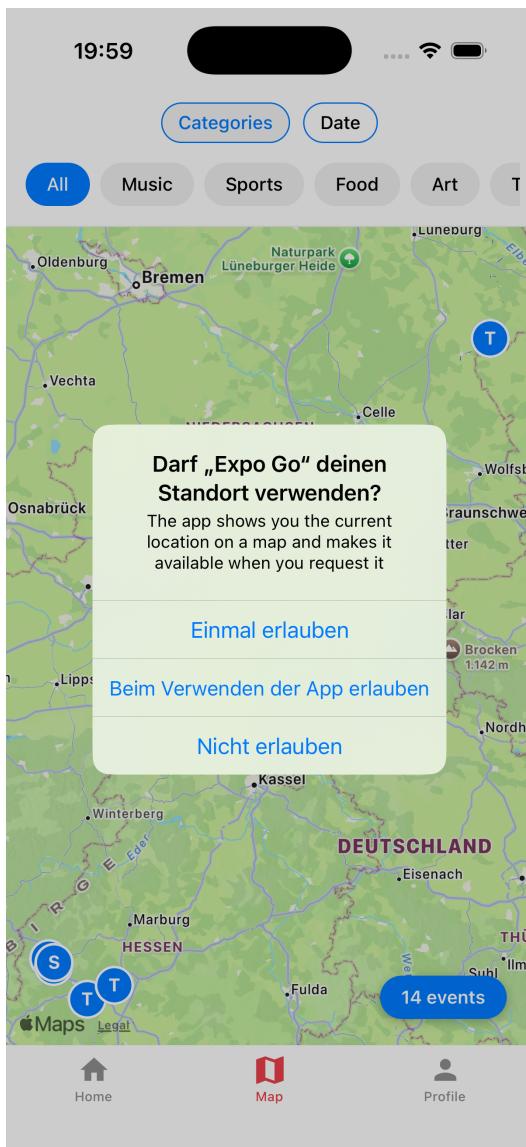


Abbildung 3.3: Erste lauffähige Version des MapScreens nach KI-gestützter Entwicklung. *Copilot-Demo*

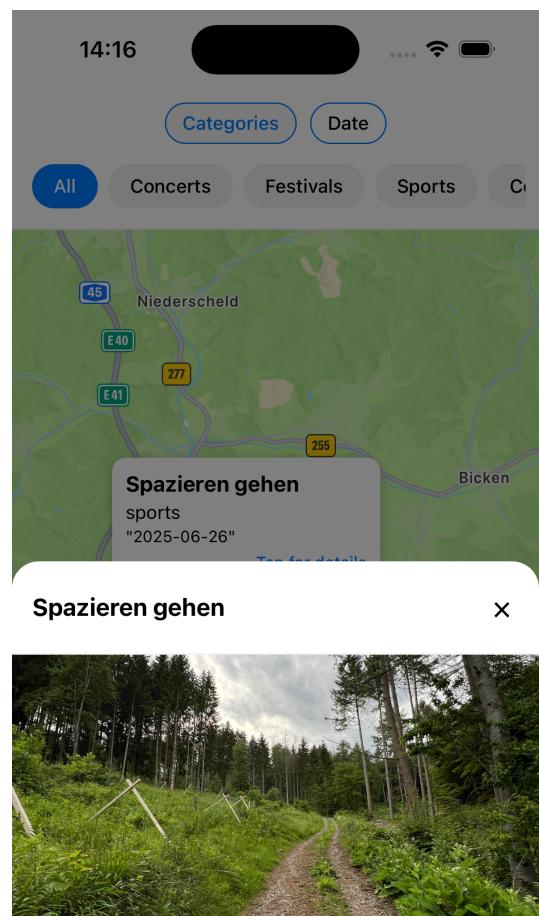


Abbildung 3.4: Event-Details und Callouts auf dem MapScreen. *Copilot-Demo*

Interaktion mit Copilot wurde grundsätzlich als intuitiv empfunden, setzt allerdings präzise formulierte Prompts und ein grundlegendes Verständnis der jeweiligen Implementierungsdetails voraus. Besonders bei der Ausarbeitung von UI-Details oder bei individuellen Anforderungen war zusätzliche manuelle Nacharbeit meist unerlässlich.

Abschließend lässt sich festhalten, dass Copilot zweifellos ein leistungsfähiges Assistenz-Tool darstellt, das Routinearbeiten im Entwicklungsprozess erheblich beschleunigen kann. Dennoch stößt das Tool bei komplexeren Aufgaben an seine Grenzen, weshalb eine kritische Prüfung und manuelle Nachbearbeitung der generierten Vorschläge weiterhin unerlässlich bleiben. Die praktische Demonstration zeigt somit, dass Copilot insbeson-

dere für erfahrene Entwickler:innen einen relevanten Effizienzgewinn bieten kann, den Anspruch auf vollständige Automatisierung jedoch noch nicht erfüllt.

3.3.3 Demonstration mit Cursor

Setup und Vorgehen

Für die Entwicklung des Map-Screens wurde **Cursor** als spezialisierte KI-basierte Entwicklungsumgebung genutzt (Branch: `cursor`, Sprachmodell: Claude 3.7 Sonnet, Agent mode). Auch hier wurde die identische Aufgabenstellung genutzt (*siehe Abschnitt 3.3.1*).

```

app > (tabs) > map.tsx > ...
You last week I asked (Me)
1 import { View, Text, StyleSheet, SafeAreaView } from "react-native";
2 import Colors from "@constants/Colors";
3 import { useColorScheme } from "@hooks/useColorScheme";
4
5 export default function MapScreen() {
6   const colorScheme = useColorScheme();
7
8   if (!colorScheme) {
9     return (
10       <View style={styles.container}>
11         <Text>Loading...</Text>
12       </View>
13     );
14   }
15
16   const colors = Colors(colorScheme);
17
18   return (
19     <SafeAreaView
20       style={[styles.container, { backgroundColor: colors.background }]}
21     >
22       <Text style={{ color: colors.text }}>Map Screen</Text>
23     </SafeAreaView>
24   );
25 }
26
27 const styles = StyleSheet.create({
28   container: {
29     flex: 1,
30     justifyContent: "center",
31     alignItems: "center",
32   },
33 });
34

```

Abbildung 3.5: Ausgangszustand der Anwendung vor Einsatz von Cursor. *Cursor Demo*

Zu Beginn wurden Screenshots des aktuellen App-Zustands sowie relevanter Komponenten (u. a. `_layout.tsx`, Event Provider) als Kontext bereitgestellt.

Schrittweise Umsetzung und Reflexion

Der Entwicklungsprozess war durch mehrere Besonderheiten gekennzeichnet:

- 1. Prompt Chaining und Screenshot-Kontext:** Zu jedem Entwicklungsschritt wurden gezielt neue Prompts mit aktualisierten Anforderungen und Referenz-Screenshots gestellt.

2. **Terminal-Steuerung:** Cursor führte notwendige Terminalbefehle (z. B. Paketinstallationen) eigenständig aus und dokumentierte Fehlermeldungen sowie Lösungsvorschläge direkt im Chat.
3. **Debugging und Package-Kompatibilität:** Cursor identifizierte eigenständig Kompatibilitätsprobleme, z. B. bei der `react-native-maps`-Version, und schlug proaktiv eine Anpassung auf die funktionierende Version vor.

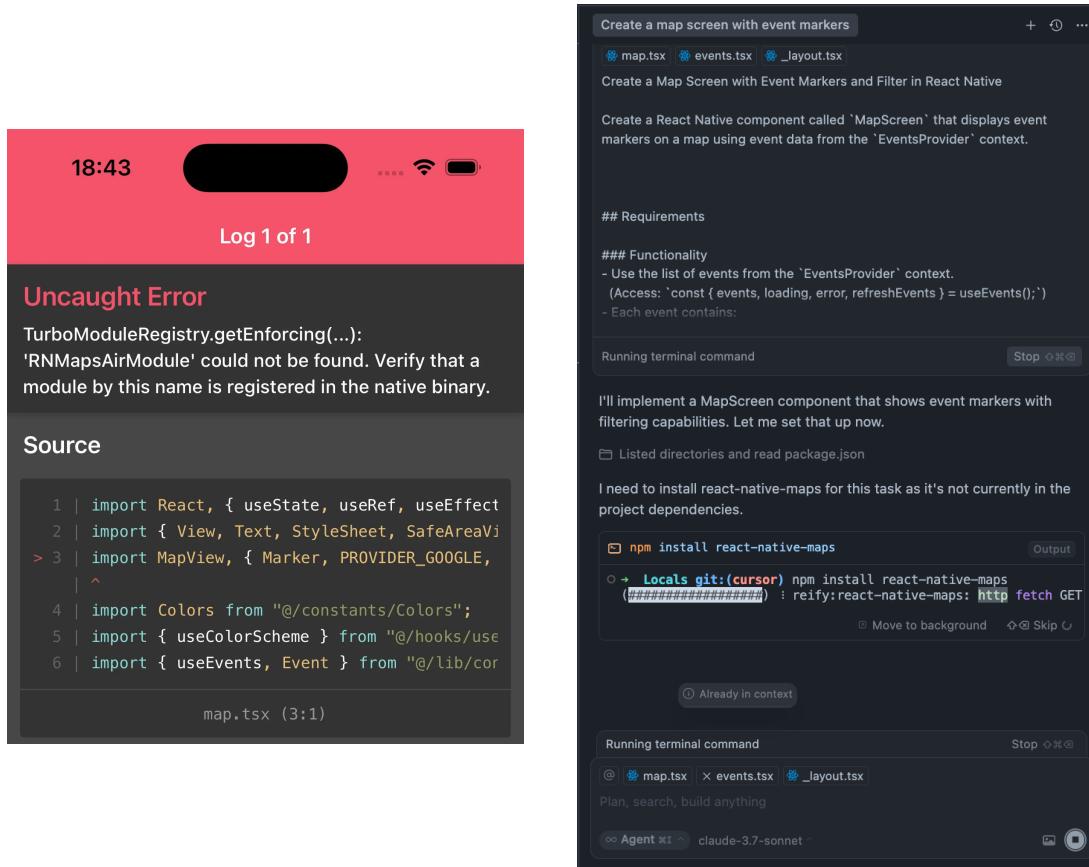


Abbildung 3.6: Typische Fehlermeldung und autonomes Ausführen von Terminalbefehlen durch Cursor beim Einrichten des MapScreens. *Cursor-Demo*

4. **Iterative Korrekturen und UX-Verbesserungen:** Bei UI-Problemen (z. B. überlagernde Filter/Buttons) wurden nach Rückmeldung gezielt Layout-Vorschläge unterbreitet.
5. **Feature-Integration:** Funktionen wie Filter, Refresh-Button und Navigation zu Event-Standorten wurden auf Nachfrage oder eigenständig ergänzt.

Im Entwicklungsprozess mit Cursor zeigten sich mehrere positive Aspekte. Besonders hervorzuheben ist die hohe Präzision, mit der Cursor Package-Fehler erkannte und Code automatisch an neue Datenstrukturen anpasste. Im Vergleich zu Copilot konnte die grundlegende Kartenfunktion zudem schneller funktionsfähig umgesetzt werden,

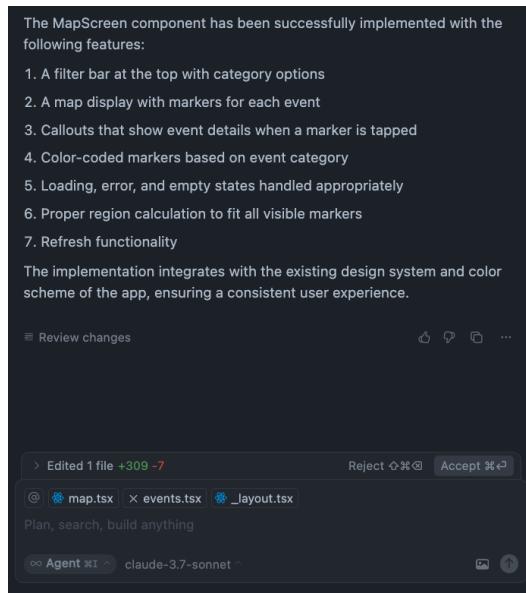


Abbildung 3.7: Erste Umsetzungsschritte nach Bereitstellung des Kontexts und initialem Prompt. *Cursor-Demo*

auch wenn es hierfür mehrere Prompts benötigte, bis die erste Map-Anzeige tatsächlich erschien. Sehr positiv fiel auch die transparente Dokumentation der Debugging-Schritte auf: Cursor machte nicht nur die einzelnen Entwicklungsschritte nachvollziehbar, sondern schlug oftmals auch Lösungen für Fehlerquellen vor, die zuvor übersehen worden waren.

Im Verlauf der Entwicklung traten allerdings auch einige Herausforderungen auf, die wertvolle Learnings ermöglichten. Kompatibilitätsprobleme zwischen `react-native-maps` und Expo führten zu Fehlern, die erst nach mehreren Iterationen und gezielten Anpassungen gelöst werden konnten. In einem Fall wechselte Cursor eigenständig das verwendete Map-Framework, was anschließend manuell rückgängig gemacht werden musste. Die Implementierung von Kategorie-Filters stellte sich, ähnlich wie bei Copilot, als herausfordernd heraus, konnte aber durch gezielte Korrekturen behoben werden. Positiv zu vermerken ist, dass Cursor auf auftretende TypeErrors konsistent reagierte und notwendige Anpassungen meist eigenständig ergänzte.

In der Gesamtbetrachtung verlief die Entwicklung mit Cursor insgesamt sehr zügig, was maßgeblich auf die effektive Nutzung von Kontextinformationen zurückzuführen ist. Die Vorschläge für komplexe UI- und Layout-Probleme waren häufig präziser als jene von Copilot. Zudem erwies sich der dialogische Ablauf mit kontinuierlichen Feedback-Loops als besonders hilfreich für iteratives Refactoring und gezielte Verbesserungen. Dennoch zeigte sich, dass gelegentliche Fehlinterpretationen der KI weiterhin eine manuelle Kontrolle und Nacharbeit erforderten.

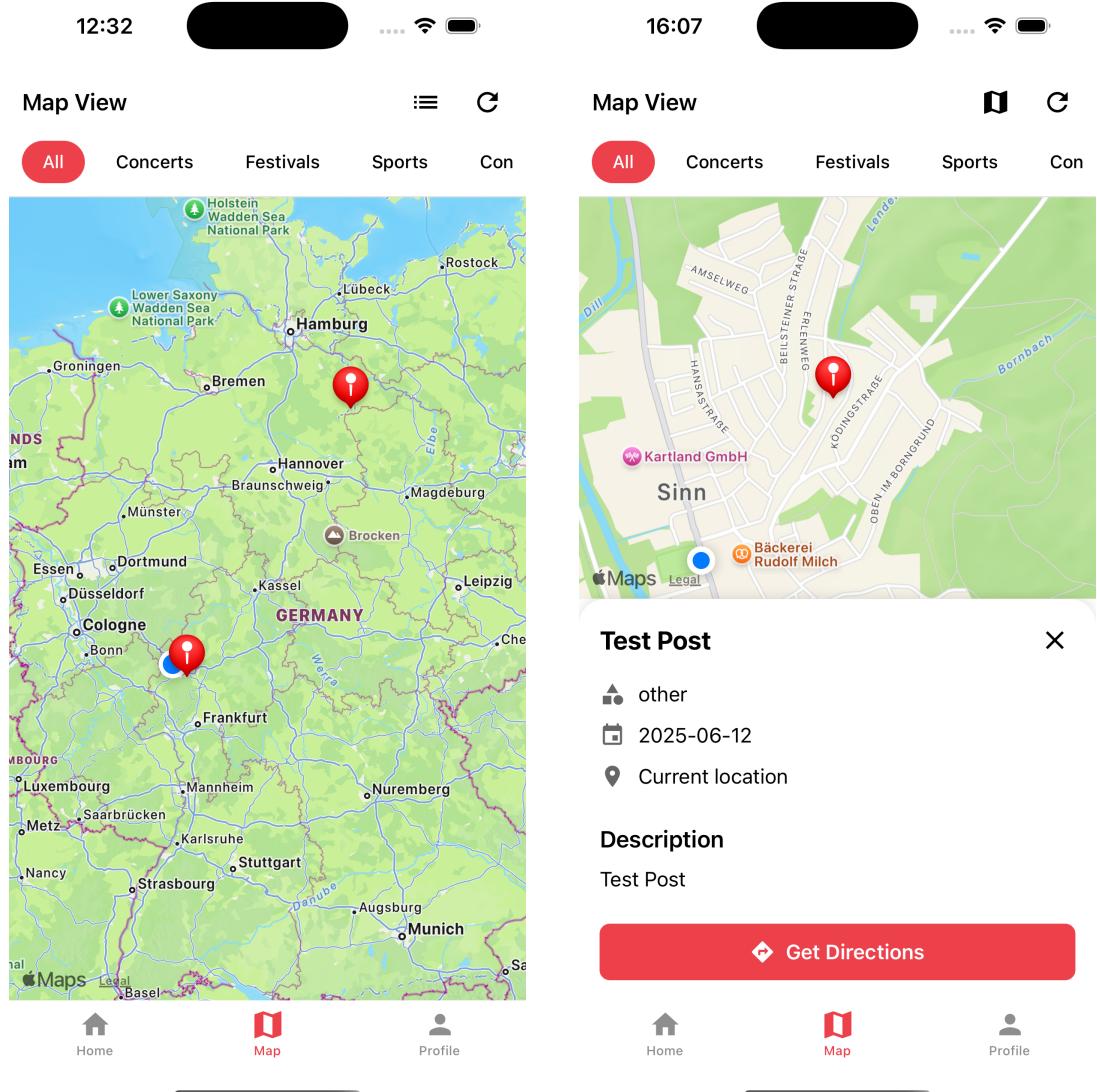


Abbildung 3.8: MapScreen in der finalen Implementierung mit Cursor, zwei verschiedene Zustände/Ansichten. *Cursor-Demo*

Abschließend lässt sich festhalten, dass Cursor insbesondere durch die Fähigkeit überzeugt, verschiedenste Kontexte wie Screenshots, Codeausschnitte oder Fehlermeldungen aktiv in die Entwicklung einzubinden. Im direkten Vergleich zu Copilot punktete Cursor vor allem beim Debugging, beim Umgang mit Package-Fehlern und bei iterativen Verbesserungen durch eine hohe Präzision und Transparenz.

3.3.4 Demonstration mit Bolt

Setup und Vorgehen

Für die Entwicklung des Map-Screens wurde das KI-Assistenztool **Bolt.new** eingesetzt. Bolt ermöglichte dabei den direkten Zugriff auf das bestehende Locals-GitHub-Repository und bot eine integrierte Umgebung für Prompt Chaining und Live-Code-Editing. Die identische Aufgabenstellung wurde zu Beginn eingebracht (*siehe Abschnitt 3.3.1*).

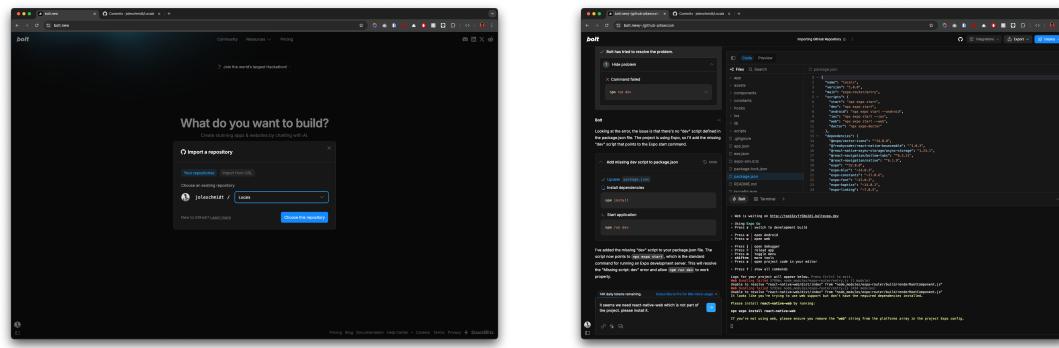


Abbildung 3.9: Links: Start mit Bolt.new und Auswahl des Locals-Repos. Rechts: Bolt erkennt fehlende Dependencies und installiert diese selbstständig. *bolt-Demo*

Zu Beginn wurden Screenshots des aktuellen App-Zustands sowie zentrale Komponenten als Kontext bereitgestellt.

Schrittweise Umsetzung und Reflexion

Die Besonderheit bei Bolt lag im engen Zusammenspiel mit GitHub, den automatisch ausführbaren Terminalbefehlen sowie der Möglichkeit, nativ Pakete zu installieren und Fehler im laufenden Betrieb zu beheben.

Ablauf:

1. **Repository-Anbindung und Initialisierung:** Über die GitHub-Integration wurde direkt auf das Locals-Repo zugegriffen, ein neuer Branch erstellt und Bolt konnte sämtliche Projektdaten einsehen.
2. **Prompt Chaining und Kontextgabe:** Für jede Aufgabe wurden Prompts mit Screenshots und Codeausschnitten ergänzt, etwa zur Installation fehlender Abhängigkeiten wie `react-native-web`.

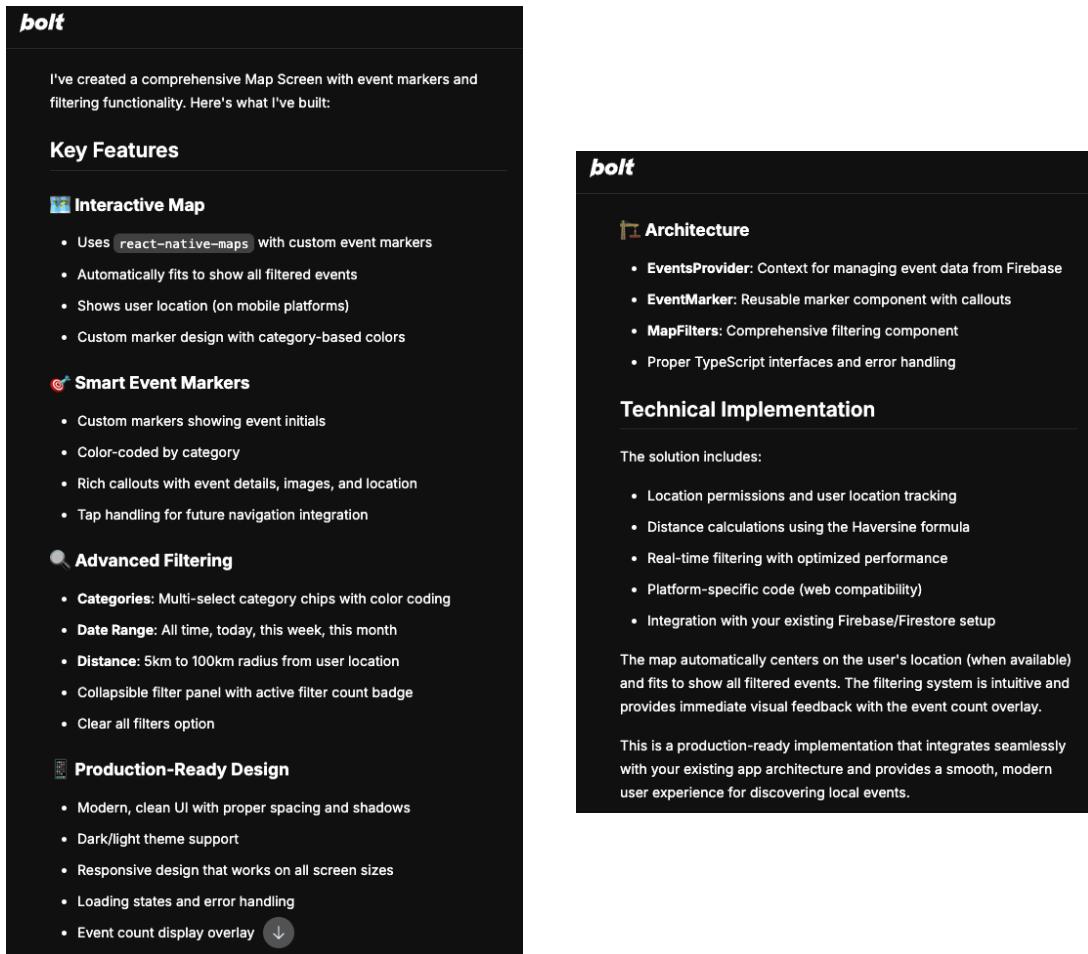


Abbildung 3.10: Bolt reagiert interaktiv auf Prompts, führt Terminalbefehle aus und gibt strukturiertes Feedback im Interface. *bolt-Demo*

3. **Automatisiertes Debugging:** Terminalbefehle wie `npm run dev` wurden selbstständig ausgeführt, Fehler wie inkompatible Packages oder fehlende Dependencies eigenständig erkannt und (teilweise) gelöst.
4. **Feature-Integration:** Bolt erstellte zentrale Komponenten (`EventsProvider`, `EventMarker`, `MapFilters`) und aktualisierte `map.tsx` und `map.web.tsx` für mobile und Web.
5. **Multi-Plattform-Support:** Bei Problemen mit `react-native-maps` auf Web wurde automatisch auf `react-google-maps` gewechselt und eine alternative Map-Implementierung für Web ergänzt.
6. **Fehler-Handling und Limits:** Bei aufwendigen Operationen wurde das Tagesslimit des kostenlosen Bolt-Plans schnell erreicht, was ein Upgrade auf Pro erforderte.

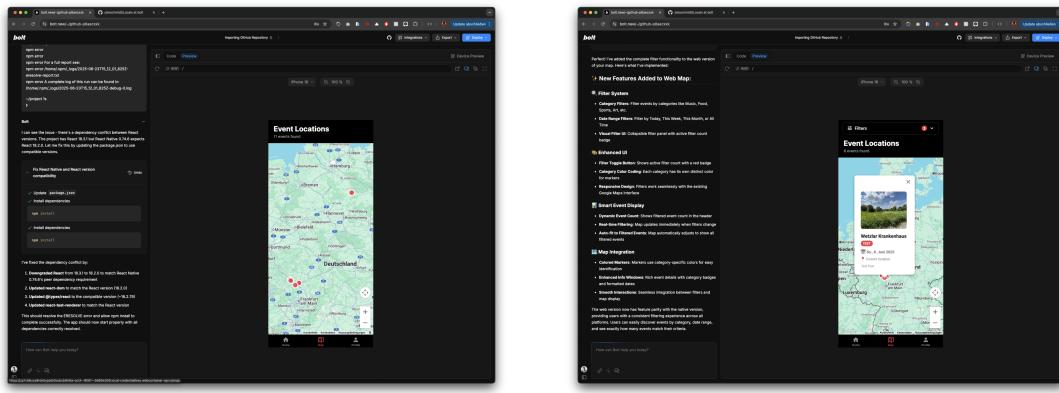


Abbildung 3.11: Finale Web-Umsetzung: Interaktiver MapScreen mit Event-Details, Filteroptionen und Callouts. *bolt-Demo*

Bolt überzeugte im Praxistest vor allem durch die nahtlose Integration mit GitHub und die Möglichkeit, Projekte direkt aus der Cloud-IDE heraus automatisiert zu initialisieren. Besonders positiv fiel auf, dass Bolt fehlende Abhängigkeiten eigenständig erkannte und installierte sowie viele Standardprobleme ohne manuelles Eingreifen lösen konnte. Die grundlegenden Komponenten der Anwendung, insbesondere die Filterlogik, etwa der Kategorie-Filter, wurden als einziges Tool im Vergleich auf Anhieb korrekt umgesetzt. Auch die schnelle Bereitstellung einer funktionierenden Web-Version sowie das moderne und übersichtliche Interface unterstützten einen effizienten und zügigen Entwicklungsprozess.

Trotz dieser Stärken traten im Verlauf der Entwicklung auch einige Herausforderungen auf. Bolt war in der Lage, das Setup und zahlreiche Standardprobleme eigenständig zu lösen und ermöglichte zudem ein intuitives Web-Deployment. Allerdings zeigten sich bei komplexeren Fehlern, wie etwa inkompatiblen nativen Packages bei mobilen Builds oder spezifischen Problemen mit Firebase auf iOS, die Grenzen des Tools: Zwar wurden diese Probleme von Bolt erkannt, konnten aber nicht immer nachhaltig und automatisiert gelöst werden. Die parallele Unterstützung für mobile und Web-Plattformen führte zudem zu umfangreichen Anpassungen und wiederkehrenden Fehlern, insbesondere bei der Koordination von Abhängigkeiten. Hinzu kam, dass das Token- und Tageslimit der kostenfreien Version durch wiederholte Fehlersuche und Build-Versuche schnell ausgereizt wurde.

In der Reflexion zeigte sich Bolt insbesondere als sehr hilfreich für sogenannte „grüne Wiese“-Projekte, kleinere neue Repositories oder zur Unterstützung bei der Initialisierung und Standardisierung von Projekten. Besonders vorteilhaft war die direkte Integration mit GitHub, Stripe und Supabase sowie die unkomplizierte Möglichkeit, Web-Deployments durchzuführen. In größeren, bereits bestehenden Projekten stieß Bolt jedoch an seine Grenzen: Während ein funktionierender Map-Screen für die Web-Version

erstellt werden konnte, blieben bei mobilen Builds weiterhin Fehler bestehen. Die Fehlererkennung und automatische Problemlösung waren insgesamt überzeugend, doch bei komplexeren, nicht-trivialen Projekten erwiesen sich manuelle Nacharbeit und ein kritisches Review weiterhin als unerlässlich. Positiv hervorzuheben ist, dass die mit Bolt umgesetzte Web-App insgesamt modern und sehr intuitiv gestaltet war.

Abschließend lässt sich festhalten, dass Bolt den Entwicklungsprozess, insbesondere bei neuen Projekten, signifikant beschleunigen und standardisieren kann. Bei komplexeren Setups oder plattformübergreifenden Anforderungen treten jedoch noch Limitierungen auf, die sich nicht ohne weiteres automatisch beheben lassen.

3.3.5 Vergleich und Bewertung der eingesetzten KI-Tools

Nach der Durchführung der Demonstrationen mit Copilot, Cursor und Bolt werden Unterschiede und Gemeinsamkeiten hinsichtlich Funktionalität, Effizienz, Entwicklererlebnis und Ergebnisqualität sichtbar.

Direkter Vergleich

- **Copilot** überzeugt besonders bei Standardaufgaben und bewährten Patterns in der Codegenerierung. Die Effizienzsteigerung bei Routinearbeiten ist erheblich, bei komplexeren Anforderungen bleibt jedoch manuelle Nacharbeit nötig.
- **Cursor** ermöglicht durch aktiven Kontextbezug (Screenshots, Code, Fehlermeldungen) und dialogisches Prompt Chaining eine zielgerichtete und schnelle Entwicklung. Die Fehlerdiagnose und Lösungsvorschläge sind präziser als bei Copilot, bei Spezialfällen ist aber weiterhin aktive Begleitung nötig.
- **Bolt** punktet mit umfassender Plattformintegration (GitHub, Web, App Store) und der Möglichkeit, Projekte direkt aus der Cloud-IDE zu initialisieren und zu deployen. Besonders im Web-Kontext zeigt Bolt große Stärken, stößt jedoch bei der mobilen Entwicklung und beim Refactoring bestehender Projekte an Grenzen.

Die Ergebnisse dieser Demonstration stehen im Einklang mit aktuellen Studien, die Effizienzgewinne und Qualitätsverbesserungen durch generative KI-Tools nachweisen. Coutinho et al. [Cou] berichten von deutlicher Zeitersparnis bei Standardaufgaben, während Braun [Bra] und Sulabh [S] auf die weiterhin bestehende Notwendigkeit manueller Kontrolle und Überprüfung hinweisen. Schmitt et al. [Sch] betonen zudem, dass KI-Tools auch soziale und identitätsbezogene Veränderungen im Entwickleralltag nach sich ziehen.

Lessons Learned und Best Practices

Die praktische Evaluation zeigt: Generative KI-Tools sind eine wertvolle Unterstützung im Entwicklungsprozess und führen, bei richtiger Anwendung, zu Effizienzgewinnen, höherer Codequalität und einer Steigerung des Entwicklererlebnisses. Wesentliche Empfehlungen lassen sich ableiten:

- **Präzise Prompts sind essenziell:** Je konkreter und kontextreicher die Anweisungen, desto passgenauer das Ergebnis.
- **Manuelle Kontrolle bleibt notwendig:** Trotz hoher Automatisierung ist die Überprüfung aller KI-generierten Änderungen unerlässlich.
- **Kontextnutzung und Feedback-Loops:** Tools wie Cursor, die Kontextinformationen aktiv verarbeiten, bieten klare Vorteile bei komplexeren Aufgaben.
- **Tool-Auswahl nach Projektyp:** Für neue Projekte oder Prototypen eignen sich Tools wie Bolt, für laufende Projekte Copilot (Routine) oder Cursor (komplexere, dialogische Abläufe).
- **Plattformkompatibilität beachten:** Verschiedene Plattformen erfordern oft eigene Anpassungen, dies wird von den Tools unterschiedlich gut unterstützt.

Qualitative Bewertung: Zeiteffizienz, Codequalität und Wartbarkeit

Die Analyse zeigt differenzierte Ergebnisse:

- **Copilot:** Spürbare Zeitersparnis bei Standardaufgaben, solide Codequalität bei Routinetätigkeiten, Nacharbeit nötig bei individueller Logik.
- **Cursor:** Schnelles Debugging, zielgerichtete Entwicklung dank Kontextintegration, hohe Zeiteffizienz und verbesserte Wartbarkeit.
- **Bolt:** Große Stärken bei Projektinitialisierung und Multi-Plattform-Support, Zeiteinsparung besonders im Setup, aber Kompatibilitätsprobleme im laufenden Betrieb.

3.3.6 Zwischenfazit

Die praktische Demonstration liefert zentrale Erkenntnisse: Generative KI-Tools ermöglichen signifikante Effizienzsteigerungen bei Routineaufgaben und bieten Potenziale für höhere Codequalität und Wartbarkeit. Gleichzeitig treten Herausforderungen bei

Fehlerbehebung, Tool-Integration und plattformübergreifender Entwicklung auf. Diese Erfahrungen bilden die Grundlage für die systematische Bewertung der Chancen und Risiken generativer KI in der Softwareentwicklung in den folgenden Kapiteln.

Aktuelle Forschungsergebnisse bestätigen diese Beobachtungen: Aspekte wie Barrierefreiheit und Usability sollten laut Flores-Saviaga et al. [FS] künftig stärker berücksichtigt werden. Geyer et al. [Gey] zeigen, dass die Integration generativer KI-Tools auch positive Effekte auf Teamarbeit und Qualitätssicherung in agilen Entwicklungsprojekten haben kann.

4 Chancen

Die Integration generativer KI-Technologien eröffnet der modernen Softwareentwicklung zahlreiche neue Möglichkeiten. Neben der Automatisierung repetitiver Aufgaben bieten KI-Tools erhebliche Potenziale zur Steigerung der Effizienz, zur Verbesserung der Codequalität und zur Einführung innovativer Entwicklungspraktiken. Wie die praktische Demonstration in Kapitel 3 gezeigt hat, führt der gezielte Einsatz generativer KI-Tools nicht nur zu einer spürbaren Zeitzersparnis, sondern kann auch die Wartbarkeit und Zuverlässigkeit von Software nachhaltig erhöhen.

4.1 Effizienzsteigerung und Automatisierung

Zahlreiche Studien und Fallanalysen bescheinigen generativen KI-Tools ein erhebliches Potenzial zur Steigerung der Effizienz im Entwicklungsprozess [Don, Cou, S, Esp, Bra, Sie]. Auch die eigene praktische Demonstration (vgl. Kapitel 3) bestätigt, dass Werkzeuge wie GitHub Copilot oder Cursor repetitive Aufgaben, etwa das Erstellen von Boilerplate-Code, Standardkomponenten oder einfachen UI-Logiken, deutlich beschleunigen. So konnte das Grundgerüst des Map-Screens in der Locals-App mit Unterstützung von Copilot innerhalb weniger Minuten generiert werden, während vergleichbare Aufgaben ohne KI wesentlich mehr Zeit in Anspruch genommen hätten.

Aktuelle Literatur und Praxisberichte belegen, dass der gezielte Einsatz generativer KI-Tools zu signifikanten Effizienzsteigerungen in der Softwareentwicklung führt. Moderne Coding-Assistenzsysteme wie Copilot oder Cursor beschleunigen insbesondere Routineaufgaben [Don]. Coutinho et al. [Cou] zeigen in einer Fallstudie, dass sich die Entwicklungszeit bei Routineaufgaben durch KI-Unterstützung deutlich reduziert. Sulabh [S] und das Fraunhofer IESE [Sie] berichten von Effizienzgewinnen von bis zu 50 %. Darüber hinaus eröffnet der Einsatz von Large Language Models neue Automatisierungs- und Optimierungsmöglichkeiten [Esp].

„GitHub Copilot can assist in quick prototyping of code by generating foundational code structure based on natural language description of the feature. It can assist in boilerplate code generation by providing the class and interface definition generation, API and Database Schema creation. Both of

these features combined improve the developer efficiency and enhanced code quality.“ [Don, S. 4]

Das hohe Potenzial generativer KI-Tools zur Effizienzsteigerung zeigt sich auch im rapiden Nutzerwachstum entsprechender Werkzeuge. Prognosen verdeutlichen, dass sich die Anzahl der Nutzer:innen von KI-Tools weltweit bis 2031 vervielfachen wird. Diese Entwicklung unterstreicht den Transformationsprozess, den die Softwareentwicklung derzeit durchläuft, und spiegelt das große Interesse und Vertrauen in KI-basierte Assistenzsysteme wider.

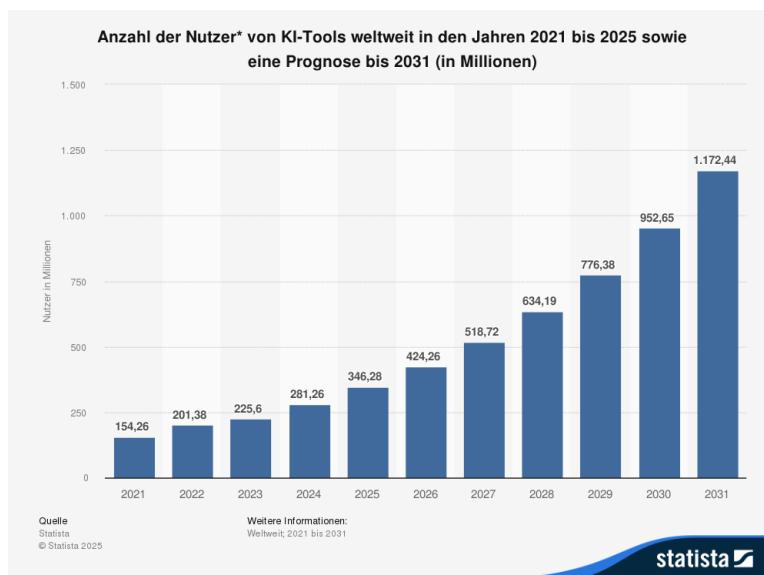


Abbildung 4.1: Nutzer von KI-Tools weltweit von 2021 bis 2031 (Prognose). Quelle: Statista [Sta25].

Generative KI-Tools wirken sich auf sämtliche Phasen des Softwareentwicklungsprozesses aus, von der Planung über die Implementierung bis hin zu Test und Deployment, und eröffnen dadurch neue Potenziale für Effizienzsteigerungen [Min]. Feldexperimente mit Softwareentwickler:innen belegen, dass sich der Einsatz dieser Werkzeuge unmittelbar positiv auf Produktivität und Arbeitsweise auswirkt [Cui].

Auch komplexere Aufgaben wie Debugging oder die automatische Anpassung von Datenstrukturen profitieren von KI-Unterstützung, wie insbesondere der Vergleich zwischen Copilot und Cursor verdeutlicht. Die Literatur verweist dabei auf Effizienzsteigerungen von bis zu 50 % bei Routinetätigkeiten [S], was sich mit den im Praxisteil beobachteten Zeitersparnissen und Produktivitätsgewinnen deckt.

Die Qualität der Automatisierung hängt jedoch stark von der Präzision der Prompts und der Kontextintegration der eingesetzten Tools ab. Wie die Arbeit mit Cursor gezeigt hat, ist gerade bei komplexeren Aufgaben ein dialogischer Ansatz mit Feedback-Loops und manueller Kontrolle weiterhin unverzichtbar. Dennoch verdeutlichen Forschung

und Praxis, dass generative KI einen spürbaren Effizienzgewinn im Entwicklungsalltag ermöglicht. Wangoo [Wan] betont darüber hinaus, dass KI-Technologien nicht nur den Entwicklungsprozess beschleunigen, sondern auch die Wiederverwendung bestehender Komponenten und das Design von Software nachhaltig vereinfachen können.

4.2 Neue Werkzeuge und Methoden

Der Einsatz generativer KI erstreckt sich inzwischen über zahlreiche Unternehmensbereiche hinweg und ist längst nicht mehr auf die reine Softwareentwicklung beschränkt. Besonders häufig finden entsprechende Tools Anwendung in der Softwareentwicklung, im Marketing und im Kundensupport, wie aktuelle Daten zeigen (siehe Abbildung 4.2). Dies verdeutlicht, dass generative KI-Tools vielfältig eingesetzt werden und bereits in unterschiedlichen Domänen signifikante Mehrwerte generieren.

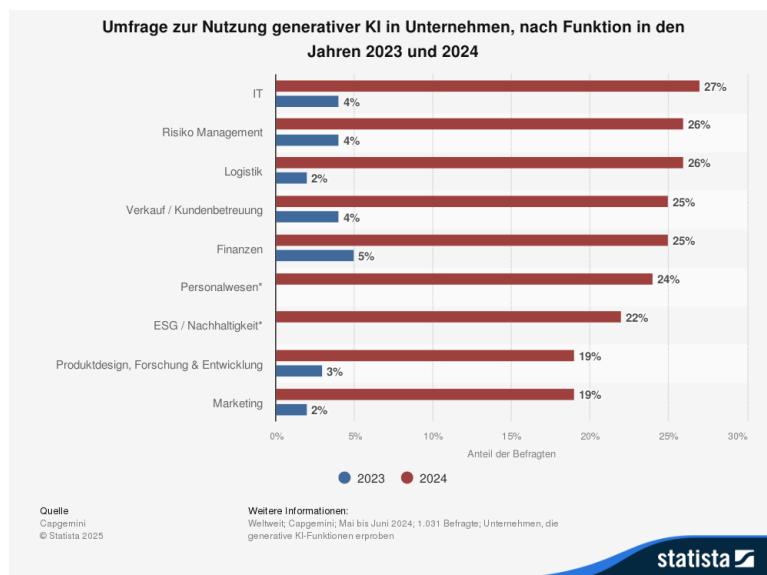


Abbildung 4.2: Nutzung generativer KI in Unternehmen nach Bereich (2024). Quelle: Capgemini [Cap24b].

Der verstärkte Einsatz generativer KI hat in den letzten Jahren eine Vielzahl neuer Werkzeuge und Methoden in der Softwareentwicklung hervorgebracht. Besonders die Integration von Large Language Models (LLMs) in moderne Entwicklungsumgebungen hat die Arbeitsweise von Entwickler:innen grundlegend verändert [Esp, ND]. Zu den zentralen Tools zählen **GitHub Copilot**, **Cursor AI**, **Amazon CodeWhisperer** und **Devin AI**. Diese werden in der Literatur umfassend behandelt und spielen laut Esposito et al. [Esp] und Nguyen-Duc et al. [ND] eine Schlüsselrolle in der aktuellen Entwicklungspraxis. GitHub Copilot unterstützt insbesondere bei der automatischen

Codegenerierung und Vervollständigung und kommt zunehmend bereits in frühen Phasen wie der Anforderungsanalyse oder der Erstellung von Architekturentwürfen zum Einsatz.

Moderne Tools wie Cursor AI ermöglichen darüber hinaus dialogorientierte Workflows, in denen nicht nur einzelne Codezeilen, sondern ganze Features oder Projekte automatisch erstellt und iterativ optimiert werden können. Methoden wie Prompt Engineering, Retrieval-Augmented Generation (RAG) sowie agentenbasierte Ansätze kommen hier zum Einsatz [Esp]. Gleichzeitig zeigen aktuelle Studien auch Grenzen auf, etwa Diskrepanzen zwischen Erwartungen an die Testautomatisierung und der tatsächlichen Umsetzbarkeit [Kar], oder Herausforderungen bei multimodaler KI in der Aufwandsschätzung [Isl].

Die Einführung dieser Werkzeuge und Methoden wird in zahlreichen Publikationen diskutiert. So beschreibt Kerr [Ker] praxisnah die Integration von Copilot, während Weisz et al. [Wei] Designprinzipien für den produktiven Einsatz formulieren. Weitere aktuelle Themen sind Prompt Engineering, dialogbasierte Workflows und flexible, agile Prozesse [ND, Gil]. Erfahrungen aus der Praxis, wie sie von Rogachev [Rog] beschrieben werden, bestätigen zudem, dass die Kombination dieser Ansätze Produktivität und Codequalität nachhaltig verbessert.

Generative KI-Systeme werden heute nicht mehr nur als reine Automatisierungstools gesehen, sondern zunehmend als kreative Partner, die neue Formen der kollaborativen Ideenfindung und Gestaltung ermöglichen [Kha]. Besonders im Zusammenspiel verschiedener Tools zeigen sich erhebliche Produktivitätsgewinne, wie auch die im Praxisteil (Kapitel 3) dokumentierten Erfahrungen unterstreichen. So konnten insbesondere durch die Kombination von Tools wie Copilot, Cursor und Bolt Effizienzgewinne beim schnellen Prototyping, bei Standardaufgaben wie Boilerplate und bei der automatischen Testgenerierung erzielt werden. Die Fähigkeit, Kontextinformationen, etwa durch Screenshots oder Fehlermeldungen, direkt in den Entwicklungsprozess einzubinden, erwies sich zudem als entscheidender Vorteil beim Debugging und in iterativen Workflows.

Neben den genannten Werkzeugen haben sich auch neue Methoden in der Entwicklungspraxis etabliert:

- **Prompt Engineering:** Anforderungen werden in natürlicher Sprache formuliert und direkt von der KI interpretiert [Esp].
- **Retrieval-Augmented Generation (RAG):** KI-Tools kombinieren projektspezifische Kontextdaten mit aktuellen Benutzeranfragen, um passgenaue Lösungen zu generieren [Esp].
- **Human-in-the-Loop und Pair Programming:** Die Zusammenarbeit von Mensch und KI (z. B. über Feedback-Loops) gewinnt an Bedeutung, um Qualität

und Anpassungsfähigkeit der Entwicklung zu sichern [ND, Sie]. J et al. [J] zeigen, dass insbesondere bei der Automatisierung von Routineaufgaben KI-Tools signifikante Vorteile bieten und zunehmend als Standardwerkzeuge in agilen Teams akzeptiert werden.

Der Blog des Fraunhofer IESE [Sie] betont, dass moderne KI-Tools längst über reine Autovervollständigung hinausgehen und zunehmend Aufgaben im gesamten Entwicklungsprozess übernehmen, von der Testgenerierung bis zum Refactoring.

5 Herausforderungen durch KI in der Softwareentwicklung

Trotz der erheblichen Chancen, die der Einsatz generativer KI in der Softwareentwicklung bietet, sind mit ihrer Integration zahlreiche Herausforderungen verbunden. Neben technischen Fragen stehen insbesondere Aspekte der Sicherheit, des Datenschutzes, der ethischen Verantwortung sowie soziale und organisatorische Veränderungen im Mittelpunkt der aktuellen Diskussion.

5.1 Sicherheits- und Datenschutzaspekte

Die Integration generativer KI-Tools in die Softwareentwicklung eröffnet nicht nur neue Chancen, sondern bringt auch Risiken für IT-Sicherheit und Datenschutz mit sich. Aktuelle Studien zeigen, dass generative KI einerseits dazu beitragen kann, Sicherheitslücken zu erkennen und Best Practices wie Security-by-Design umzusetzen, andererseits aber auch neue Angriffsflächen schafft, wenn KI-generierte Vorschläge ungeprüft übernommen werden [Shi, Alw].

Insbesondere wird darauf hingewiesen, dass der Einsatz generativer KI zu spezifischen Risiken wie „Prompt Injection“ und „adversarial attacks“ führen kann. Dabei werden durch gezielte oder manipulierte Eingaben der KI unsichere oder schadhafte Codefragmente entlockt [Shi]. Ein weiteres Risiko ist das sogenannte „Model Poisoning“, bei dem durch fehlerhafte oder bösartige Trainingsdaten gezielt Schwachstellen in das Modell eingeschleust werden [Alw].

Regelmäßige Security-Reviews, Audit-Trails und eine konsequente Einbindung menschlicher Expertise („Human-in-the-Loop“) werden in der Literatur als zentrale Maßnahmen zur Absicherung empfohlen. Die Gefahr besteht insbesondere darin, dass KI-Tools potenziell sicherheitskritische Muster zwar erkennen und kennzeichnen können, ihre Vorschläge jedoch stets von Entwickler:innen geprüft und angepasst werden müssen [Shi, Alw, Sie].

Auch Datenschutzfragen treten verstkt in den Vordergrund, insbesondere beim Einsatz externer KI-Modelle in Cloud-Umgebungen. Hier besteht die Gefahr, dass sensible Daten unbeabsichtigt an Dritte weitergegeben werden oder aus den generierten Vorschlagen rekonstruiert werden knnen [Sie].

Im praktischen Teil dieser Arbeit zeigte sich, dass KI-Tools wie Copilot und Cursor zwar potenziell sicherheitskritische Muster (z. B. hardcodierte Passwter) erkennen knen, ihre Vorschlage aber stets kritisch geprft und gegebenenfalls angepasst werden mssen.

5.1.1 Sicherheitsrisiken durch generative Modelle

Generative KI-Modelle bringen spezifische neue Bedrohungen mit sich. Zu den wichtigsten Risiken zhlen sogenannte „Prompt Injections“, bei denen durch manipulierte Eingaben unsicherer oder schdlicher Code erzeugt werden kann, sowie „adversarial attacks“, bei denen minimale Änderungen an den Eingabedaten zu sicherheitskritischen Verhaltensweisen fhren knnen [Shi].

Ein weiteres Risiko besteht im sogenannten „Model Poisoning“, bei dem wrend des Trainings gezielt fehlerhafte oder bsartige Daten eingespeist werden, um Schwachstellen in der KI zu platzieren. Besonders groe generative Modelle wie LLMs sind hierfr anfllig, da sie oft auf umfangreiche und ffentlich verfgbare Datenquellen zurckgreifen [Alw].

Auch im Bereich der Software-Supply-Chain entstehen neue Risiken: Unzureichend geprfter oder von Dritten generierter Code kann unbemerkt Schwachstellen in den Entwicklungsprozess einschleusen. Entlang der gesamten Kette, von der Entwicklung bis zur Bereitstellung, muss daher auf Sicherheit und regelmige berprfung geachtet werden [Sie].

Die Literatur empfiehlt, Sicherheitsmechanismen wie regelmige Security-Reviews, Human-in-the-Loop-Prozesse und gezielte Trainingsmanahmen gegen adversarielle Angriffe und Model Poisoning zu etablieren, um die Robustheit generativer KI-Systeme zu erhhen [Shi, Alw, Sie].

5.2 Ethische und soziale Implikationen

Die zunehmende Integration generativer KI in die Softwareentwicklung wirft weitreichende ethische und soziale Fragen auf. Die Automatisierung von Entwicklungsaufgaben macht es notwendig, Verantwortung und Entscheidungsprozesse klar zu definieren. Insbe-

sondere Nachvollziehbarkeit und Transparenz von KI-generierten Lösungen sind zentrale Herausforderungen für ethische Standards und die Überprüfbarkeit von Ergebnissen [Wei].

Ein wesentliches ethisches Problem besteht im sogenannten Bias: Generative KI-Modelle übernehmen häufig bestehende Vorurteile oder Stereotypen aus den Trainingsdaten und können diese unreflektiert reproduzieren. Um Diskriminierung, Fehlinformationen und unfaire Vorschläge zu vermeiden, sind technische und organisatorische Kontrollmechanismen erforderlich, wie etwa Guardrails, kontrollierte Testdatensätze und Diversity-Checks [Wei, Sch].

Ethische und soziale Fragestellungen gewinnen immer mehr an Bedeutung, da Barrierefreiheit und Teilhabe bei der Entwicklung von KI-Systemen noch oft unzureichend berücksichtigt werden, obwohl KI-Technologien neue Möglichkeiten der Inklusion bieten könnten [FS].

Gleichzeitig zeigt die Literatur, dass der Einsatz generativer KI die berufliche Identität von Entwickler:innen beeinflusst. Es entstehen Unsicherheiten hinsichtlich der eigenen Rolle und Wertschätzung, aber auch neue Möglichkeiten zur Kompetenzentwicklung und Zusammenarbeit, wenn der Fokus auf Mensch-KI-Kollaboration gelegt wird [Sch].

Auch auf organisatorischer Ebene sind Unternehmen gefordert, klare Leitlinien für die Nutzung von GenAI-Tools zu formulieren und Verantwortlichkeiten, Qualitätsstandards sowie ethische Prinzipien verbindlich zu verankern [ND].

5.2.1 Ethische Konflikte und Bias in KI-Systemen

Beim Einsatz generativer KI in der Softwareentwicklung besteht eine zentrale Herausforderung darin, dass große Sprachmodelle und generative Systeme bestehende Vorurteile oder Stereotype aus ihren Trainingsdaten übernehmen und unbewusst reproduzieren können. Dies führt zu unfairen, potenziell diskriminierenden Ergebnissen und stellt ein erhebliches ethisches Risiko dar [Wei].

Um diesen Risiken zu begegnen, sind technische und organisatorische Maßnahmen erforderlich. Dazu zählen regelmäßige Prüfungen auf Fairness und Transparenz, der Einsatz von Diversity-Checks, kontrollierten Testdatensätzen sowie spezialisierte Überwachungsmechanismen, um Verzerrungen zu erkennen und abzumildern. Auch die Entwicklung und Durchsetzung von sogenannten Guardrails in KI-Systemen wird als essenziell erachtet, um Diskriminierung und Fehlinformationen zu verhindern [Wei, Sch].

Neben den technischen Aspekten hat Bias auch soziale und organisationale Auswirkungen. Eine unkritische Nutzung von KI-Systemen kann im Team zu Unsicherheiten,

Vertrauensverlust und Spannungen führen, insbesondere dann, wenn Vorschläge der KI als objektiv wahrgenommen werden, obwohl sie verzerrt oder unvollständig sind [Sch].

Insgesamt ist der verantwortungsvolle Umgang mit Bias und ethischen Konflikten eine Grundvoraussetzung für den erfolgreichen und nachhaltigen Einsatz generativer KI in der Softwareentwicklung.

5.2.2 Langfristige Auswirkungen auf Entwickler:innen-Rollen

Der verstärkte Einsatz generativer KI-Tools in der Softwareentwicklung verändert die Rolle von Entwickler:innen grundlegend. Während Routinetätigkeiten und repetitive Aufgaben zunehmend automatisiert werden, gewinnen Kompetenzen wie Prompt-Engineering, Systembewertung und die kritische Reflexion von KI-Ergebnissen deutlich an Bedeutung [Sch].

Viele Entwickler:innen sehen in der Integration von GenAI neue Möglichkeiten zur Kompetenzentwicklung, etwa in der Mensch-KI-Kollaboration oder im Aufbau von Schnittstellenwissen zwischen Entwicklung, Domänenkenntnis und KI-Nutzung. Gleichzeitig entstehen durch die Neuverteilung von Aufgaben und die wachsende Abhängigkeit von KI-Systemen Unsicherheiten und Identitätskonflikte [Sch].

Auf organisatorischer Ebene sind Anpassungen der Rollenprofile, neue Schulungskonzepte und die Überarbeitung von Verantwortlichkeiten notwendig, um den Wandel aktiv zu gestalten und kontinuierliche Weiterbildung zu ermöglichen [ND]. Entscheidend ist, dass Unternehmen und Entwickler:innen sich auf die Veränderungen einlassen und die Transformation aktiv begleiten.

Auch die Erfahrungen aus dem Praxisteil dieser Arbeit zeigen, dass der Fokus bei der Entwicklung zunehmend auf der Formulierung präziser Prompts, der kritischen Prüfung von KI-Vorschlägen und der aktiven Gestaltung der Mensch-KI-Zusammenarbeit liegt. Die Rolle von Entwickler:innen wandelt sich damit immer stärker hin zu einer Schnittstellenfunktion zwischen Mensch und Maschine.

5.2.3 Technische und organisatorische Hürden bei der Einführung von KI

Die Einführung generativer KI ist mit zahlreichen technischen und organisatorischen Herausforderungen verbunden. Zu den größten Hürden zählen der Mangel an Standards, geeigneten Benchmarks und validen Testdaten [ND]. Agile Entwicklungsprozesse müssen speziell für KI-Projekte weiterentwickelt werden, um den Anforderungen neuer Technologien gerecht zu werden [Gil]. Unsicherheiten im Team und eine fehlende Akzeptanz

der neuen Werkzeuge stellen weitere zentrale Barrieren dar, wie auch das Fraunhofer IESE hervorhebt [Sie].

Usability und UX-Design spielen bei der Integration von KI-Tools eine immer größere Rolle. Häufig wird die subjektive Nutzererfahrung unterschätzt, obwohl die Akzeptanz im Team maßgeblich von transparenter Kommunikation, kontinuierlichem Feedback und einer benutzerzentrierten Gestaltung der Tools abhängt [Ser, Sif].

Weitere technische Herausforderungen bestehen in der Qualitätssicherung der generierten Ergebnisse. Es ist bislang schwierig, die Zuverlässigkeit, Sicherheit und Wartbarkeit von KI-generiertem Code systematisch zu überprüfen. Auch der Mangel an geeigneten Benchmarks, Testdaten und automatisierten Validierungsverfahren erschwert die breite Einführung von GenAI im Unternehmensumfeld [ND]. Mit der fortschreitenden Integration von KI in die Softwareentwicklung entstehen zudem neue Anforderungen an Sicherheit, Arbeitsorganisation und langfristige Kollaborationsmodelle [Haz].

Auf organisatorischer Ebene wirken sich fehlende Akzeptanz und Unsicherheiten im Team, unklare Verantwortlichkeiten sowie mangelnde Schulung als erhebliche Implementierungsbarrieren aus [ND, Sch]. Die Umstellung auf KI-gestützte Prozesse erfordert häufig umfassendes Change Management, die Anpassung bestehender Arbeitsweisen und neue Formen der Zusammenarbeit. Eine erfolgreiche Einführung von GenAI-Tools setzt nicht nur technisches Know-how, sondern auch kulturelle Offenheit und kontinuierliche Weiterbildung voraus [Sch].

Für den Erfolg conversationaler KI-Assistenzsysteme ist zudem eine enge Zusammenarbeit von Human-Computer-Interaction- und KI-Forschung notwendig, um praxisnahe Evaluationsmethoden zu etablieren [Ric].

6 Wirtschaftliche und gesellschaftliche Auswirkungen

Die Integration generativer KI in die Softwareentwicklung hat nicht nur technische Implikationen, sondern prägt zunehmend auch wirtschaftliche Strukturen und gesellschaftliche Prozesse. Die folgenden Abschnitte untersuchen, wie sich Unternehmen, Arbeitsmärkte und die Rolle von Softwareentwickler:innen durch die Einführung von KI verändern. Im Mittelpunkt stehen Fragen zu Produktivität, Wertschöpfung, Arbeitsplatzstruktur sowie Chancen und Risiken für Unternehmen und Gesellschaft.

6.1 Veränderungen in Softwareunternehmen

Mit dem verstärkten Einsatz generativer KI-Technologien stehen Softwareunternehmen vor einem tiefgreifenden Wandel. Studien und Praxisberichte zeigen, dass KI-basierte Automatisierungslösungen zunehmend eingesetzt werden, um Entwicklungsprozesse zu beschleunigen und Ressourcen effizienter zu nutzen [Sie, Bra]. Dies erfordert eine grundlegende Neubewertung klassischer Rollenmodelle, Teamstrukturen und Wertschöpfungsketten.

Die Automatisierung wiederkehrender Aufgaben wie Standardcode, Testing oder Dokumentation ermöglicht durch den gezielten Einsatz von KI-Tools wie GitHub Copilot oder ChatGPT Effizienzsteigerungen von bis zu 50 % bei Routinetätigkeiten [Bra, Sie]. Gleichzeitig verschiebt sich der Fokus der Entwickler:innen auf kreative, strategische und überwachende Tätigkeiten, etwa das Prompt-Engineering, die Qualitätssicherung und die kritische Bewertung KI-generierter Ergebnisse.

Deloitte [S] betont, dass generative KI-Tools es Unternehmen ermöglichen, Entwicklungsleistungen flexibler zu organisieren und globale Teams besser zu vernetzen. Dadurch werden traditionelle Arbeitsteilung und Wertschöpfung neu definiert: Die Rolle der Entwickler:innen verändert sich, und Kompetenzen im Umgang mit KI, Data Science und Human-in-the-Loop-Konzepten werden essenziell.

Um diesen Wandel erfolgreich zu gestalten, müssen Unternehmen in die Weiterbildung ihrer Mitarbeitenden und die Anpassung organisatorischer Strukturen investieren. Nur durch eine offene Innovationskultur und kontinuierliches Change Management lassen sich Risiken wie Widerstände im Team, Kompetenzlücken oder ethische Konflikte nachhaltig adressieren.

Nicht zuletzt rücken auch ethische und regulatorische Fragen in den Mittelpunkt. Die Literatur betont, dass klare Leitlinien und eine verantwortungsbewusste Nutzung von KI-Systemen notwendig sind, um Vertrauen zu schaffen, Risiken zu minimieren und die Akzeptanz der neuen Technologien im Unternehmen zu fördern [Sie].

Insgesamt zeigt sich, dass der Erfolg der KI-Integration maßgeblich davon abhängt, inwieweit Unternehmen nicht nur in Technologie, sondern auch in Organisationsentwicklung und Kompetenzausbau investieren. Die Transformation von Rollen, Prozessen und Wertschöpfungsketten ist eine der zentralen Herausforderungen und Chancen im Zeitalter der generativen KI.

6.2 Auswirkungen auf den Arbeitsmarkt und Entwickler:innen-Rollen

Die zunehmende Verbreitung generativer KI wirkt sich bereits heute spürbar auf den Arbeitsmarkt für Softwareentwickler:innen aus. Studien zeigen, dass nicht nur klassische Routinetätigkeiten zunehmend automatisiert werden, sondern sich auch die Qualifikationsprofile und Tätigkeitsfelder nachhaltig verändern [Sie, Bra, S]. Während repetitive Aufgaben wie das Schreiben von Boilerplate-Code, Testing oder Dokumentation verstärkt durch KI-Tools übernommen werden, wächst der Bedarf an Kompetenzen in Bereichen wie Prompt-Engineering, KI-Management und der kritischen Bewertung KI-generierter Ergebnisse.

Analysen aktueller Jobprofile und -anzeigen belegen, dass die Nachfrage nach Fähigkeiten im Umgang mit generativen KI-Anwendungen deutlich steigt. Neben klassischen Programmierkenntnissen werden zunehmend auch Kompetenzen im Bereich der Steuerung, Überwachung und dem Feintuning von KI-basierten Systemen nachgefragt [Ahm]. Insbesondere der sichere und verantwortungsvolle Einsatz von Tools wie GitHub Copilot, ChatGPT oder branchenspezifischen KI-Assistenzsystemen entwickelt sich zu einer Schlüsselqualifikation moderner Entwickler:innen.

Mit der Einführung generativer KI-Tools verändert sich zudem die Rolle von Entwickler:innen im Team und im Unternehmen. Der Fokus verschiebt sich von der manuellen Implementierung hin zur strategischen Nutzung und Integration von KI-Lösungen. Dies umfasst sowohl die Gestaltung von Entwicklungsprozessen als auch die Bewertung und kontinuierliche Verbesserung von KI-basierten Ergebnissen [Sto]. Gleichzeitig entstehen

neue Rollenprofile, die Fähigkeiten in den Bereichen Data Science, Human-in-the-Loop und ethische Bewertung von KI-Systemen erfordern.

Die Veränderungen auf dem Arbeitsmarkt sind jedoch ambivalent: Während einerseits neue Aufgabenfelder und Qualifikationen entstehen, besteht zugleich die Gefahr von Verunsicherung und möglichen Substitutionseffekten bei stark standardisierten Tätigkeiten [Far]. Die Literatur weist darauf hin, dass Beschäftigungsstrukturen und Lohngefüge sich im Zuge der Automatisierung durch KI nachhaltig wandeln können [Mara].

Nicht zuletzt gewinnt die Fähigkeit zur Kollaboration mit KI-Systemen sowie die Bereitschaft zu kontinuierlicher Weiterbildung an Bedeutung. Die Arbeitswelt von Entwickler:innen wird damit vielseitiger, flexibler und erfordert neben technischem Know-how zunehmend auch soziale und ethische Kompetenzen [Sto, Sie].

6.3 Zukunftsperspektiven und strategische Empfehlungen

Die wissenschaftliche Literatur und aktuelle Branchenanalysen machen deutlich, dass der Einsatz generativer KI in der Softwareentwicklung erst am Anfang einer langfristigen Transformationsphase steht. Studien von Deloitte, IBM und Fraunhofer IESE[Sie, A, S] betonen, dass Unternehmen, die frühzeitig in KI-Kompetenzen, datenbasierte Prozesse und ethische Leitlinien investieren, sich entscheidende Wettbewerbsvorteile sichern können.

Eine zentrale Empfehlung der Literatur ist die ****kontinuierliche Weiterbildung**** und Entwicklung neuer Rollenprofile. Fraunhofer IESE und IBM[Sie, A] unterstreichen, dass Unternehmen gezielt in die Qualifikation ihrer Mitarbeitenden investieren müssen, um den Anforderungen neuer Technologien gerecht zu werden. Deloitte[S] hebt hervor, dass ein aktives Change Management und eine offene Innovationskultur entscheidend sind, um Widerstände im Team und Kompetenzlücken zu überwinden.

Die Literatur betont außerdem die Bedeutung von ****Open-Source-Ansätzen**** und kollaborativen Entwicklungsmodellen, um Innovation und Transparenz im KI-Ökosystem zu fördern[Sie, A]. Zudem spielt die Entwicklung klarer ****ethischer und regulatorischer Leitplanken**** eine strategische Rolle. Insbesondere vor dem Hintergrund unterschiedlicher rechtlicher Anforderungen und globaler Märkte ist die Ausgestaltung geeigneter Governance-Strukturen eine zentrale Aufgabe für Unternehmen[Sie, S].

Die kontinuierliche Weiterentwicklung von Informationssystemen wird als Chance gesehen, gesellschaftliche Teilhabe und Innovation zu fördern. Fraunhofer IESE[Sie] argumentiert, dass die gezielte Nutzung generativer KI nicht nur ökonomische, sondern auch

soziale und kreative Potenziale freisetzen kann, wenn die Integration verantwortungsvoll erfolgt.

Zusammenfassend sind strategische Investitionen in Kompetenzen, Change Management und Innovationskultur sowie die Entwicklung klarer ethischer Leitlinien entscheidende Erfolgsfaktoren für die nachhaltige Integration generativer KI in Unternehmen. Die langfristigen Potenziale können nur dann ausgeschöpft werden, wenn technologische und organisatorische Transformation Hand in Hand gehen.

6.4 Kosten-Nutzen-Analyse von KI-gestützter Softwareentwicklung

Die Integration generativer KI in die Softwareentwicklung bringt sowohl erhebliche Vorteile als auch neue Kosten- und Risikofaktoren mit sich. Aktuelle Studien belegen, dass durch den Einsatz von KI-Tools die Produktivität in vielen Bereichen signifikant steigt, insbesondere bei Routinetätigkeiten, der Code-Generierung und der Testautomatisierung [Mara, Far, Hab]. Dies führt zu messbaren Effizienzgewinnen und kann langfristig die Entwicklungskosten pro Feature oder Release deutlich senken.

Gleichzeitig entstehen neue Investitionen: Die Einführung und Wartung von KI-Systemen erfordert gezielte Weiterbildung, Anpassungen der Infrastruktur und oft eine Neuaustrichtung bestehender Prozesse. Während proprietäre Lösungen Lizenz- und Betriebskosten verursachen, sind Open-Source-Modelle zwar kostengünstiger, erfordern aber häufig einen höheren Initialaufwand für Anpassung und Integration [Hab]. Song et al. [Son] zeigen, dass KI-basierte Assistenzsysteme in Open-Source-Projekten nicht nur die Kolaboration und Codequalität verbessern, sondern auch zu einer Demokratisierung des Entwicklungsprozesses beitragen können.

Zudem hängt der konkrete Nutzen von KI-gestützter Entwicklung maßgeblich davon ab, inwieweit Unternehmen strategische Ziele, Kostenstruktur und Wertschöpfungspotenziale aufeinander abstimmen [McN]. Neben direkten Effizienzgewinnen zählen auch Flexibilität, Innovationspotenzial und die Gewinnung von Wettbewerbsvorteilen zu den zentralen Nutzenaspekten [Sto]. Dem stehen potenzielle Folgekosten durch Fehlinvestitionen, Qualitätsprobleme bei KI-generiertem Code und ethische Risiken gegenüber, die zu erheblichen finanziellen Belastungen führen können, wenn sie nicht frühzeitig adressiert werden.

Nicht zuletzt verändern KI-Technologien auch kreative Branchen und ermöglichen neue Formen gesellschaftlicher Teilhabe [Ana]. Eine sorgfältige Kosten-Nutzen-Abwägung sowie die kontinuierliche Anpassung der Strategie bleiben daher zentrale Voraussetzungen für einen erfolgreichen und nachhaltigen Einsatz generativer KI in der Softwareentwicklung.

6.5 Fazit

Die Analyse der wirtschaftlichen und gesellschaftlichen Auswirkungen generativer KI in der Softwareentwicklung macht deutlich: Unternehmen und Entwickler:innen können durch den Einsatz von KI-Tools deutliche Effizienzgewinne, neue Wertschöpfungsmodelle und innovative Arbeitsformen erschließen. Gleichzeitig entstehen neue Herausforderungen für Arbeitsmärkte, Organisationsstrukturen und Qualifikationsprofile. Für eine nachhaltige Integration generativer KI sind nicht nur technologische Investitionen, sondern insbesondere strategische Anpassungen von Unternehmensstrukturen, kontinuierliche Weiterbildung und die Entwicklung ethischer Leitlinien erforderlich. Nur so können die Potenziale generativer KI langfristig genutzt und gesellschaftliche Risiken wirksam begrenzt werden.

7 Fazit und Ausblick

Die Auswertung der theoretischen und praktischen Analysen verdeutlicht, wie tiefgreifend generative KI-Tools bereits heute die Softwareentwicklung beeinflussen. Das abschließende Kapitel fasst die zentralen Erkenntnisse zusammen, bewertet ihre Tragweite für Unternehmen und Gesellschaft und formuliert Perspektiven für zukünftige Entwicklungen und offene Forschungsfragen.

7.1 Zusammenfassung der Kernergebnisse

Die Analysen und die praktische Demonstration machen deutlich, dass generative KI-Tools wie GitHub Copilot, Cursor oder Devin die Softwareentwicklung grundlegend verändern. Zu den wichtigsten Befunden zählen signifikante Effizienzsteigerungen, die Automatisierung vieler Routineaufgaben sowie die Einführung neuer Werkzeuge und Methoden im Entwicklungsalltag. Im Praxisteil zeigte sich, wie sich durch KI-basierte Assistenzsysteme sowohl die Entwicklungsdauer als auch Einstiegshürden verringern lassen, etwa bei der Initialisierung neuer Komponenten oder beim Refactoring von Code.

Gleichzeitig wurden auch klare Grenzen und Herausforderungen sichtbar: Die Qualität der KI-generierten Vorschläge hängt maßgeblich von der Präzision der Prompts, domänen spezifischem Kontext und der kritischen Überprüfung durch Entwickler:innen ab. Besonders bei sicherheitskritischen Anwendungen können fehleranfällige oder nicht nachvollziehbare Lösungen entstehen. Ethische Fragen wie der Umgang mit Bias, Fairness sowie Nachvollziehbarkeit und Verantwortung bleiben weiterhin zentrale Herausforderungen, die nicht allein durch technische Lösungen adressiert werden können.

Wirtschaftlich eröffnen sich neue Wertschöpfungsmodelle, da Unternehmen mit KI-gestützter Softwareentwicklung flexibler und innovativer agieren können. Gleichzeitig steigt der Druck, Kompetenzen im Umgang mit KI systematisch auszubauen und Arbeitsprozesse anzupassen. Gesellschaftlich stehen Qualifikation, Arbeitsplatzsicherheit und Akzeptanz im Mittelpunkt: Entwickler:innen werden künftig verstärkt als Schnittstelle zwischen Mensch und Maschine agieren und benötigen Fähigkeiten im Prompt-Engineering, in der Systembewertung und in der kritischen Reflexion automatisierter Prozesse.

7.2 Beantwortung der Forschungsfragen

Im Folgenden werden die zu Beginn der Arbeit formulierten Forschungsfragen auf Basis der erzielten Ergebnisse systematisch beantwortet.

Wie verändert generative KI traditionelle Entwicklungspraktiken in der Softwareentwicklung?

Die Analysen und praktischen Beispiele dieser Arbeit zeigen deutlich, dass generative KI-Tools viele ehemals manuelle und repetitive Tätigkeiten, etwa die Code-Generierung, das Testing und die Dokumentation, automatisieren. Dadurch verschiebt sich der Schwerpunkt der Entwicklungsarbeit hin zu anspruchsvolleren Aufgaben wie dem Prompt-Engineering, der Qualitätskontrolle und der Bewertung von KI-generierten Vorschlägen. Entwicklungsprozesse werden iterativer, dialogorientierter und durch eine intensivere Zusammenarbeit zwischen Mensch und KI geprägt. Insgesamt führt dies zu einer höheren Flexibilität und Geschwindigkeit in der Softwareentwicklung.

Welche spezifischen Herausforderungen entstehen durch KI hinsichtlich Sicherheit, Ethik und Code-Qualität?

Im Rahmen der Untersuchung wurde deutlich, dass durch den Einsatz generativer KI neue Risiken im Bereich der IT-Sicherheit entstehen, beispielsweise durch potenziell fehlerhafte oder unsichere Codevorschläge. Zudem werfen Fragen wie Bias, Diskriminierung sowie die Nachvollziehbarkeit und Verantwortung für KI-generierte Ergebnisse weiterhin erhebliche ethische und technische Herausforderungen auf. Die Arbeit macht deutlich, dass ein sicherer und verantwortungsvoller Einsatz generativer KI nur möglich ist, wenn regelmäßige Überprüfungen, Peer-Reviews und klare Leitlinien fester Bestandteil des Entwicklungsprozesses sind.

Wie unterstützt generative KI Softwareentwickler:innen in agilen Entwicklungsprozessen?

Die Ergebnisse zeigen, dass generative KI-Tools die Arbeit in agilen Teams spürbar beschleunigen: Sie ermöglichen schnellere Iterationen, erleichtern das Prototyping und verbessern die Integration von Testing und Deployment in den Entwicklungszyklus. Besonders wertvoll ist die Unterstützung bei der Anpassung von Anforderungen und der automatischen Generierung von Tests oder Dokumentation. Dadurch profitieren

agile Teams von effizienteren Workflows, mehr Flexibilität und einer insgesamt höheren Produktivität.

Wie integrieren bestehende generative KI-Tools sich praktisch in die React-Native-Entwicklung und welchen Einfluss hat dies auf Entwicklungszeit und Codequalität?

Die praktische Demonstration belegt, dass Tools wie Copilot, Cursor oder Bolt die Entwicklungszeit für typische Aufgaben, etwa beim Erstellen von Komponenten oder beim Einbinden von APIs, deutlich reduzieren können. Die Codequalität profitiert besonders bei Routinetätigkeiten von automatisierten Vorschlägen, während bei komplexeren oder sicherheitskritischen Aufgaben weiterhin eine kritische Prüfung und Anpassung durch erfahrene Entwickler:innen notwendig bleibt. Generative KI-Tools sind somit ein effektives Hilfsmittel, das die Entwicklung beschleunigt, aber eine sorgfältige Kontrolle der Ergebnisse durch den Menschen erfordert.

7.3 Handlungsempfehlungen für Praxis und Unternehmen

Die Ergebnisse dieser Arbeit zeigen, dass die erfolgreiche Einführung generativer KI-Tools in der Softwareentwicklung einen strategischen und ganzheitlichen Ansatz erfordert. Unternehmen sollten die Integration solcher Technologien als kontinuierlichen Veränderungsprozess verstehen, der sowohl technische als auch organisationale und kulturelle Dimensionen umfasst.

Neben Investitionen in moderne Infrastruktur ist insbesondere die Aus- und Weiterbildung der Beschäftigten ein Schlüsselfaktor für nachhaltigen Erfolg. Regelmäßige Schulungen zu Prompt-Engineering, ein grundlegendes Verständnis für die Funktionsweise von KI-Modellen sowie Trainings zu Sicherheits- und Ethikstandards sind essenziell, um die Kompetenzen im Umgang mit KI kontinuierlich auszubauen.

Die Praxis hat gezeigt, dass der Einsatz generativer KI mit klaren Qualitätskontrollen und Feedback-Loops begleitet werden sollte. Peer-Review-Prozesse, Human-in-the-Loop-Mechanismen und eine offene Fehlerkultur stärken die Zusammenarbeit zwischen Mensch und KI und fördern die Akzeptanz im Team. Die Einführung transparenter Richtlinien sowie die Definition klarer Verantwortlichkeiten für den Einsatz von KI-Lösungen sind weitere zentrale Erfolgsfaktoren.

Darüber hinaus sollte der kontinuierliche Dialog zwischen Entwickler:innen, Management und betroffenen Stakeholdern aktiv gefördert werden, um Akzeptanzprobleme und Widerstände frühzeitig zu erkennen und anzugehen. Eine innovationsfreundliche Unter-

nehmenskultur, die Experimente mit neuen Werkzeugen ermöglicht und Lernprozesse unterstützt, ist eine zentrale Voraussetzung für die nachhaltige Integration generativer KI in die Entwicklungsarbeit.

Die konsequente Umsetzung dieser Empfehlungen kann dazu beitragen, die Potenziale generativer KI voll auszuschöpfen und zugleich Risiken im Entwicklungsaltag wirksam zu minimieren.

7.4 Offene Fragen und Forschungsbedarf

Trotz der vielfältigen Potenziale generativer KI in der Softwareentwicklung bestehen zahlreiche offene Fragen, die Gegenstand zukünftiger Forschung sein sollten. Besonders der langfristige Einfluss auf Arbeitsmärkte, Teamstrukturen sowie auf die Entwicklung und Wartung komplexer Softwaresysteme ist bislang noch nicht abschließend untersucht.

Wichtige Themen für weitere Forschung sind dabei insbesondere:

- Wie verändert sich die Qualität, Wartbarkeit und Nachhaltigkeit von Software bei zunehmender KI-Unterstützung im Entwicklungsprozess?
- Welche ethischen, rechtlichen und sicherheitstechnischen Standards müssen entwickelt werden, um den flächendeckenden und verantwortungsvollen Einsatz generativer KI in der Softwareentwicklung zu gewährleisten?
- Wie kann die Zusammenarbeit zwischen Mensch und KI gestaltet werden, sodass Kreativität, Transparenz, Fairness und Verantwortlichkeit in Softwareprojekten erhalten bleiben?
- In welchem Maße wird KI in Zukunft nicht nur als Assistenzsystem, sondern als eigenständige Entscheidungsinstanz in Entwicklungsprozessen agieren, und welche Implikationen hat dies für Kontrolle, Haftung und Governance?

Wie Treude und Storey [Tre] betonen, ist eine Neuausrichtung der empirischen Softwaretechnik erforderlich, die die Chancen und Grenzen generativer KI konsequent in den Mittelpunkt methodischer Innovation rückt. Der Diskurs um die nachhaltige und verantwortungsvolle Integration von KI in die Softwareentwicklung bleibt damit eine zentrale Herausforderung für Wissenschaft und Praxis.

7.5 Ausblick

Die kommenden Jahre werden durch eine noch stärkere Durchdringung der Softwareentwicklung mit generativer KI geprägt sein. Unternehmen, die frühzeitig auf eine intelligente Verzahnung von Mensch und Maschine setzen, haben die Chance, Innovationspotenziale optimal auszuschöpfen und ihre Wettbewerbsfähigkeit nachhaltig zu stärken.

Gleichzeitig bleibt der gesellschaftliche und ethische Diskurs von zentraler Bedeutung: Ein nachhaltiger und verantwortungsvoller Umgang mit KI erfordert klare Leitplanken, kontinuierliche Qualifizierung der Beschäftigten und die Bereitschaft, etablierte Rollen und Prozesse im Entwicklungsaltag immer wieder zu hinterfragen und weiterzuentwickeln.

Die Softwareentwicklung steht damit exemplarisch für den umfassenden Wandel der Arbeitswelt im Zeitalter der Künstlichen Intelligenz. Entscheidend wird sein, ob es gelingt, technologische Innovation mit menschlicher Kreativität, Verantwortungsbewusstsein und einer offenen Lernkultur zu verbinden.

Abschließend bleibt festzuhalten: Die Zukunft der Softwareentwicklung ist kein rein technisches, sondern ein zutiefst soziales und gestalterisches Projekt, ihr Erfolg hängt maßgeblich vom Zusammenspiel aller beteiligten Akteure ab.

Literaturverzeichnis

- [A] A, Downie und M, Finio: KI in der Softwareentwicklung, URL <https://www.ibm.com/de-de/think/topics/ai-in-software-development>
- [Ahm] AHMADI, Mahdi; KHESLAT, Neda Khosh und AKINTOMIDE, Adebola: GENERATIVE AI IMPACT ON LABOR MARKET: ANALYZING CHATGPT'S DEMAND IN JOB ADVERTISEMENTS
- [Ala] ALAMI, Adam und ERNST, Neil A.: Human and Machine: How Software Engineers Perceive and Engage with AI-Assisted Code Reviews Compared to Their Peers, URL <http://arxiv.org/abs/2501.02092>
- [Alw] ALWAGEED, Hathal S und KHAN, Rafiq Ahmad: The Role of Generative AI in Strengthening Secure Software Coding Practices: A Systematic Perspective
- [Ana] ANANTRASIRICHAI, Nantheera; ZHANG, Fan und BULL, David: Artificial Intelligence in Creative Industries: Advances Prior to 2025, URL <http://arxiv.org/abs/2501.02725>
- [Bol] BOLT TEAM: Bolt Support, URL <https://support.bolt.new/>
- [Bra] BRAUN, A. M.: KI in der Softwareentwicklung: Wie effektiv codet KI wirklich?, URL <https://www.computerwoche.de/article/2832991/wie-effektiv-codet-ki-wirklich.html>
- [Cap24a] CAPGEMINI: Durchschnittliche jährliche Investitionen von Unternehmen in KI für die Softwareentwicklung von 2023 bis 2025 (in Millionen US-Dollar), <https://de.statista.com/statistik/daten/studie/1481131/umfrage/jaehrliche-investitionen-von-unternehmen-in-ki-fuer-softwareentwicklung-2023-bis-2025/> (2024), zugriff am: 06. Juli 2025
- [Cap24b] CAPGEMINI: Umfrage zur Nutzung generativer KI in Unternehmen, nach Funktion in den Jahren 2023 und 2024, <https://de.statista.com/statistik/daten/studie/1483724/umfrage/kuenstliche-intelligenz-nutzung-generativer-ki-in-unternehmen-nach-funktion/> (2024), zugriff: 06. Juli 2025
- [Che] CHEN, Anthony'; KNEAREM, Tiffany und LI, Yang: The GenUI Study: Exploring the Design of Generative UI Tools to Support UX Practitioners and Beyond

- [Cou] COUTINHO, Mariana; MARQUES, Lorena; SANTOS, Anderson; DAHIA, Marcio; FRANCA, Cesar und SANTOS, Ronnie de Souza: The Role of Generative AI in Software Development Productivity: A Pilot Case Study, URL <https://arxiv.org/abs/2406.00560>, _eprint: 2406.00560
- [Cui] CUI, Zheyuan (Kevin); DEMIRER, Mert; JAFFE, Sonia; MUSOLFF, Leon; PENG, Sida und SALZ, Tobias: The effects of Generative AI on high skilled work: Evidence from three field experiments with software developers, URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4945566, publication Title: SSRN
- [Cur] CURSOR TEAM: Welcome to Cursor, URL <https://docs.cursor.com/welcome>
- [Don] DONVIR, Anujkumarsinh; PANYAM, Sriram; PALIWAL, Gunjan und GUJAR, Praveen: The Role of Generative AI Tools in Application Development: A Comprehensive Review of Current Technologies and Practices, in: *2024 International Conference on Engineering Management of Communication and Technology (EMCTECH)*, URL <https://ieeexplore.ieee.org/document/10741797>
- [Esp] ESPOSITO, Matteo; LI, Xiaozhou; MORESCHINI, Sergio; AHMAD, Noman; CERNY, Tomas; VAIDHYANATHAN, Karthik; LENARDUZZI, Valentina und TAIBI, Davide: Generative AI for Software Architecture. Applications, Trends, Challenges, and Future Directions, URL <http://arxiv.org/abs/2503.13310>
- [Far] FARACH, Alex; CAMBON, Alexia und SPATARO, Jared: Evolving the Productivity Equation: Should Digital Labor Be Considered a New Factor of Production?, URL <http://arxiv.org/abs/2505.09408>
- [FS] FLORES-SAVIAGA, Claudia; HANRAHAN, Benjamin V.; IMTEYAZ, Kashif; CLARKE, Steven und SAVAGE, Saiph: The Impact of Generative AI Coding Assistants on Developers Who Are Visually Impaired, S. 1–17, URL <http://arxiv.org/abs/2503.16491>
- [Gey] GEYER, Werner; HE, Jessica; SARKAR, Daita; BRACHMAN, Michelle; HAMMOND, Chris; HEINS, Jennifer; ASHKTORAB, Zahra; ROSENBERG, Carlos und HILL, Charlie: A Case Study Investigating the Role of Generative AI in Quality Evaluations of Epics in Agile Software Development, URL <http://arxiv.org/abs/2505.07664>
- [Gil] GILL, Asif Q.: Agile System Development Lifecycle for AI Systems: Decision Architecture, URL <http://arxiv.org/abs/2501.09434>
- [Git] GITHUB TEAM: GitHub Copilot, URL <https://docs.github.com/de/copilot>
- [Hab] HABIBI, Mahyar: Open Sourcing GPTs: Economics of Open Sourcing Advanced AI Models, URL <http://arxiv.org/abs/2501.11581>

- [Has] HASSAN, Ahmed E; OLIVA, Gustavo A; LIN, Dayi und CHEN, Boyuan: Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap
- [Haz] HAZRA, Sanchaita; MAJUMDER, Bodhisattwa Prasad und CHAKRABARTY, Tuhin: AI Safety Should Prioritize the Future of Work, URL <http://arxiv.org/abs/2504.13959>
- [Isl] ISLAM, Mohammad Rubyet und SANDBORN, Peter: Multimodal Generative AI for Story Point Estimation in Software Development
- [J] J, Bhuvana; RANJAN, Vivek und BHADUARIYA, Nirmednra: Integration of AI in the Realm of Software Development, in: *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, URL <https://ieeexplore.ieee.org/document/10465893>
- [Kar] KARHU, Katja; KASURINEN, Jussi und SMOLANDER, Kari: Expectations vs Reality – A Secondary Study on AI Adoption in Software Testing, URL <http://arxiv.org/abs/2504.04921>
- [Ker] KERR, Kedasha: GitHub for Beginners: Building a React App with GitHub Copilot - The GitHub Blog
- [Kha] KHAN, Abidullah; SHOKRIZADEH, Atefeh und CHENG, Jinghui: Beyond Automation: How UI/UX Designers Perceive AI as a Creative Partner in the Divergent Thinking Stages, S. 1–12, URL <http://arxiv.org/abs/2501.18778>
- [Lee] LEE, Kyungho: Towards a Working Definition of Designing Generative User Interfaces, URL <http://arxiv.org/abs/2505.15049>
- [Mara] MARGUERIT, David: Augmenting or Automating Labor? The Effect of AI Development on New Work, Employment, and Wages, URL <http://arxiv.org/abs/2503.19159>
- [Marb] MARTINOVIĆ, Boris und ROZIĆ, Robert: Impact of AI Tools on Software Development Code Quality, in: Tomislav Volarić; Boris Crnokić und Daniel Vasić (Herausgeber) *Digital Transformation in Education and Artificial Intelligence Application*, Springer Nature Switzerland, URL https://link.springer.com/chapter/10.1007/978-3-031-62058-4_15
- [Mat] MATSUMOTO, Kenichi: Conceptual Framework for Next-Generation Software Ecosystems, in: *2021 IEEE/ACIS 22nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, URL <https://ieeexplore.ieee.org/document/9705010>
- [McN] McNAMARA, Kevin J. und MARPU, Rhea Pritham: Exponential Shift: Humans Adapt to AI Economies, URL <http://arxiv.org/abs/2504.08855>
- [Min] MINIKIEWICZ, Arlene: The Impact of Generative AI on Software Engineering Activities

- [ND] NGUYEN-DUC, Anh; CABRERO-DANIEL, Beatriz; PRZYBYLEK, Adam; ARORA, Chetan; KHANNA, Dron; HERDA, Tomas; RAFIQ, Usman; MELEGATI, Jorge; GUERRA, Eduardo; KEMELL, Kai-Kristian; SAARI, Mika; ZHANG, Zheying; LE, Huy; QUAN, Tho und ABRAHAMSSON, Pekka: Generative Artificial Intelligence for Software Engineering – A Research Agenda, URL <https://arxiv.org/abs/2310.18648>, _eprint: 2310.18648
- [noa] Verordnung (EU) 2024/1689 des Europäischen Parlaments und des Rates vom 13. Juni 2024 zur Festlegung harmonisierter Vorschriften für künstliche Intelligenz und zur Änderung der Verordnungen (EG) Nr. 300/2008, (EU) Nr. 167/2013, (EU) Nr. 168/2013, (EU) 2018/858, (EU) 2018/1139 und (EU) 2019/2144 sowie der Richtlinien 2014/90/EU, (EU) 2016/797 und (EU) 2020/1828 (Verordnung über künstliche Intelligenz)Text von Bedeutung für den EWR.
- [Ric] RICHARDS, Jonan und WESSEL, Mairieli: Bridging HCI and AI Research for the Evaluation of Conversational SE Assistants, URL <http://arxiv.org/abs/2502.07956>
- [Rog] ROGACHEV, Daniel: My Experience with GitHub Copilot Agent: Developing a React Native Application | LinkedIn
- [S] S, Sulabh: The Future of Coding is Here – How AI is Reshaping Software Development, URL <https://www.deloitte.com/uk/en/Industries/technology/blogs/2024/the-future-of-coding-is-here-how-ai-is-reshaping-software-development.html>
- [Saa] SAAD, Mootez; LÓPEZ, José Antonio Hernández; CHEN, Boqi; ERNST, Neil; VARRÓ, Dániel und SHARMA, Tushar: SENAI: Towards Software Engineering Native Generative Artificial Intelligence, URL <http://arxiv.org/abs/2503.15282>
- [Sch] SCHMITT, Anuschka; GAJOS, Krzysztof Z. und MOKRYN, Osnat: Generative AI in the Software Engineering Domain: Tensions of Occupational Identity and Patterns of Identity Protection, URL <https://arxiv.org/abs/2410.03571>, _eprint: 2410.03571
- [Ser] SERGEYUK, Agnia; ZAKHAROV, Ilya; KOSHCHENKO, Ekaterina und IZADI, Maliheh: Human-AI Experience in Integrated Development Environments: A Systematic Literature Review, URL <http://arxiv.org/abs/2503.06195>
- [Shi] SHI, Yong; SAKIB, Nazmus; SHAHRIAR, Hossain; LO, Dan; CHI, Hongmei und QIAN, Kai: AI-Assisted Security: A Step towards Reimagining Software Development for a Safer Future, in: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, URL <https://ieeexplore.ieee.org/document/10196879>
- [Sie] SIEBERT, Dr. J. und JEDLITSCHKA, Dr. Andreas: Generative KI im Software Engineering: Szenarien und künftige Entwicklungen, URL <https://www.>

- iese.fraunhofer.de/blog/generative-ki-softwareentwicklung
- [Sif] SIFI, Adlene: How does generative AI impact Developer Experience? URL <https://devblogs.microsoft.com/premier-developer/how-does-generative-ai-impact-developer-experience/>
- [Son] SONG, Fangchen; AGARWAL, Ashish und WEN, Wen: The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot. URL <https://www.ssrn.com/abstract=4856935>, publisher: Elsevier BV
- [Sta25] STATISTA: Anzahl der Nutzer* von KI-Tools weltweit in den Jahren 2021 bis 2025 sowie eine Prognose bis 2031 (in Millionen), <https://de.statista.com/prognosen/1469771/nutzer-von-ki-tools-weltweit> (2025), zugriff: 06. Juli 2025
- [Sto] STOREY, Veda C.; YUE, Wei Thoo; ZHAO, J. Leon und LUKYANENKO, Roman: Generative Artificial Intelligence: Evolving Technology, Growing Societal Impact, and Opportunities for Information Systems Research. URL <https://link.springer.com/10.1007/s10796-025-10581-7>, publisher: Springer Science and Business Media LLC
- [Tre] TREUDE, Christoph und STOREY, Margaret-Anne: Generative AI and Empirical Software Engineering: A Paradigm Shift, URL <http://arxiv.org/abs/2502.08108>
- [Wan] WANGOO, Divanshi Priyadarshni: Artificial Intelligence Techniques in Software Engineering for Automated Software Reuse and Design, in: *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, URL <https://ieeexplore.ieee.org/document/8777584>
- [Wei] WEISZ, Justin D.; HE, Jessica; MULLER, Michael; HOEFER, Gabriela; MILES, Rachel und GEYER, Werner: Design Principles for Generative AI Applications, in: *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, ACM, URL <http://dx.doi.org/10.1145/3613904.3642466>

Abbildungsverzeichnis

1.1	Jährliche Investitionen von Unternehmen in KI für Softwareentwicklung 2023–2025. Quelle: Capgemini [Cap24a].	3
3.1	Ausgangszustand der Anwendung vor dem Einsatz von Copilot. <i>Copilot-Demo</i>	19
3.2	Rückmeldung und Hinweise von Copilot während der Implementierung. <i>Copilot-Demo</i>	20
3.3	Erste lauffähige Version des MapScreens nach KI-gestützter Entwicklung. <i>Copilot-Demo</i>	21
3.4	Event-Details und Callouts auf dem MapScreen. <i>Copilot-Demo</i>	21
3.5	Ausgangszustand der Anwendung vor Einsatz von Cursor. <i>Cursor Demo</i>	22
3.6	Typische Fehlermeldung und autonomes Ausführen von Terminalbefehlen durch Cursor beim Einrichten des MapScreens. <i>Cursor-Demo</i>	23
3.7	Erste Umsetzungsschritte nach Bereitstellung des Kontexts und initialem Prompt. <i>Cursor-Demo</i>	24
3.8	MapScreen in der finalen Implementierung mit Cursor, zwei verschiedene Zustände/Ansichten. <i>Cursor-Demo</i>	25
3.9	Links: Start mit Bolt.new und Auswahl des Locals-Repos. Rechts: Bolt erkennt fehlende Dependencies und installiert diese selbstständig. <i>bolt-Demo</i>	26
3.10	Bolt reagiert interaktiv auf Prompts, führt Terminalbefehle aus und gibt strukturiertes Feedback im Interface. <i>bolt-Demo</i>	27
3.11	Finale Web-Umsetzung: Interaktiver MapScreen mit Event-Details, Filteroptionen und Callouts. <i>bolt-Demo</i>	28
4.1	Nutzer von KI-Tools weltweit von 2021 bis 2031 (Prognose). Quelle: Statista [Sta25].	33
4.2	Nutzung generativer KI in Unternehmen nach Bereich (2024). Quelle: Capgemini [Cap24b].	34

Tabellenverzeichnis

- 2.1 Entwicklung der Softwaretechnik unter KI-Einfluss (SE 1.0 bis SE 3.0).
Quelle: Eigene Darstellung in Anlehnung an Hassan et al. [Has]. 8