

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

2. 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 2. 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	3
1.3 Methodik	4
1.4 Abgrenzung	5
2 Theoretische Grundlagen	6
2.1 Künstliche Intelligenz: Definitionen und Technologien	6
2.1.1 Beispiele für generative KI-Tools in der Praxis	7
2.1.2 Wichtige Algorithmen und Modelle in der Softwareentwicklung .	9
2.2 Generative KI-Tools: Funktion und Anwendung	11
2.2.1 Grundfunktion generativer KI-Tools	11
2.2.2 Schnittstellen und Integration in Entwicklungsumgebungen .	11
2.2.3 Beispielhafte Workflows: Pair Programming mit Copilot . . .	12
2.2.4 Vorteile und Optimierungspotenziale	12
2.2.5 Grenzen und typische Fehlerquellen	13
2.2.6 Designprinzipien für den produktiven und sicheren Einsatz .	13
3 Praktische Demonstration	15
3.1 Zielsetzung und Vorgehen	15
3.2 Vorstellung der App „Locals“	17
3.2.1 Architektur und Aufbau	17
3.2.2 Bestehende Funktionalitäten	18
3.3 Implementierung der interaktiven Kartenansicht mit KI-Unterstützung .	19
3.3.1 Prompt-Setup: Einheitliche Aufgabenstellung für alle Tools . .	19
3.3.2 Demonstration mit GitHub Copilot	20
3.3.3 Demonstration mit Cursor	24
3.3.4 Demonstration mit Bolt	28
3.3.5 Vergleich und Bewertung der eingesetzten KI-Tools	31
3.3.6 Zwischenfazit	32
4 Chancen	34
4.1 Effizienzsteigerung und Automatisierung	34
4.2 Neue Werkzeuge und Methoden	35

5 Herausforderungen durch KI in der Softwareentwicklung	38
5.1 Sicherheits- und Datenschutzaspekte	38
5.1.1 Sicherheitsrisiken durch generative Modelle	39
5.2 Ethische und soziale Implikationen	40
5.2.1 Ethische Konflikte und Bias in KI-Systemen	41
5.2.2 Langfristige Auswirkungen auf Entwickler:innen-Rollen	41
5.2.3 Technische und organisatorische Hürden bei der Einführung von KI	42
6 Wirtschaftliche und gesellschaftliche Auswirkungen	44
6.1 Veränderungen in Softwareunternehmen	44
6.2 Auswirkungen auf den Arbeitsmarkt und Entwickler:innen-Rollen	45
6.3 Zukunftsperspektiven und strategische Empfehlungen	46
6.4 Kosten-Nutzen-Analyse von KI-gestützter Softwareentwicklung	47
6.5 Fazit	48
7 Fazit und Ausblick	49
7.1 Zusammenfassung der Kernergebnisse	49
7.2 Beantwortung der Forschungsfragen	50
7.3 Handlungsempfehlungen für Praxis und Unternehmen	50
7.4 Einschränkungen der Arbeit	51
7.5 Offene Fragen und Forschungsbedarf	51
7.6 Ausblick	52
Literaturverzeichnis	53
Abkürzungsverzeichnis	58
Abbildungsverzeichnis	58
Tabellenverzeichnis	59
Listings	60
A Anhang 1	61
B Anhang 2	62

1 Einführung

Künstliche Intelligenz (KI) hat in den vergangenen Jahren einen rasanten Aufschwung erlebt und beeinflusst bereits vielfältige Branchen von der Medizin bis zur Finanzwelt. Auch die Softwareentwicklung bleibt nicht verschont: Dort eröffnen KI-gestützte Verfahren ein breites Spektrum neuer Einsatzfelder. So kann KI nicht nur das Schreiben von Code und das Durchführen automatisierter Tests erleichtern, sondern auch innovative Methoden für Fehlersuche und Qualitätssicherung bereitstellen.

Diese Potenziale gehen jedoch mit weitreichenden Fragestellungen einher. Neben technischen Aspekten wie Sicherheit und Code-Qualität spielt auch die gesellschaftliche Dimension eine Rolle, etwa die Frage nach ethischen Standards oder Veränderungen im Berufsbild „Softwareentwickler“. Insbesondere generative KI, zum Beispiel in Form von Large Language Models (LLMs), wirft Fragen zu Datenschutz, Verantwortung und methodischer Einbindung in agile Prozesse auf.

Vor diesem Hintergrund setzt die vorliegende Arbeit an: Sie soll beleuchten, wie KI-gestützte Technologien den Softwareentwicklungsprozess langfristig prägen und welche Herausforderungen sich dabei ergeben. Dies betrifft sowohl die konkrete Arbeitssituation von Entwickler:innen als auch die strategischen Überlegungen von Unternehmen.

1.1 Motivation

Die Relevanz des Themas ergibt sich aus den gegenwärtigen Entwicklungen in Forschung und Praxis: Immer mehr Unternehmen erforschen aktiv den Einsatz von KI-Technologien, um sich Effizienzvorteile und Innovationsschübe zu sichern. Gleichzeitig zeigt sich in vielen Studien ein Spannungsverhältnis zwischen den Versprechen generativer KI – zum Beispiel automatisierte Code-Generierung und intelligente Projektsteuerung – und den Risiken, etwa unzureichender Transparenz, Sicherheitslücken oder ethischen Verzerrungen.

Nach aktuellen Schätzungen (Stand: 2025) erreicht der Softwaremarkt in Deutschland ein Volumen von rund 31 Milliarden US-Dollar und Entwicklerinnen und Entwickler verbringen laut Erhebungen durchschnittlich bis zu 17 Stunden pro Woche mit

Wartungsaufgaben – dies zeigt, wie dringend Automatisierungsansätze und Qualitätsverbesserungen in der Praxis benötigt werden. Gerade hier setzt die generative KI an: Sie kann beispielsweise durch das automatisierte Erstellen von Boilerplate-Code oder durch Code-Assistenten wie GitHub Copilot nicht nur die Effizienz im Entwicklungsprozess deutlich steigern, sondern auch das Fachkräftethema ein Stück weit abfedern. Allerdings lassen sich aus diesem Trend auch kontroverse Fragen ableiten, etwa inwiefern die starke Abhängigkeit von KI-Modellen die Rollen und Kompetenzen von Softwareentwicklerinnen und -entwicklern langfristig verändert – oder wie Unternehmen sicherstellen können, dass durch KI-gestützte Automatisierung weiterhin qualitativ hochwertige, wartbare und sichere Software entsteht [Sie].

Hinzu kommt, dass Softwareentwicklung durch agile Methoden wie Scrum oder Kanban bereits stark dynamisiert ist: Teams agieren flexibel, stehen aber auch unter stetigem Veränderungsdruck. Wenn dann zusätzlich KI als Tool oder „Co-Entwickler“ eingebunden wird, steigen die Anforderungen an Prozessgestaltung, Rollenverteilung und Qualitätsmanagement weiter. Genau hier setzt diese Arbeit an: Sie möchte klären, wie Entwickler und Entscheider KI sinnvoll in den Softwarelebenszyklus integrieren können, wo praxisnahe Chancen liegen und welche neuen Stolpersteine zu beachten sind.

Dabei deuten aktuelle Marktanalysen darauf hin, dass KI-Assistenten die Arbeitsweise von Softwareentwicklern erheblich beschleunigen können. Laut einer von Deloitte zitierten Studie ist es beispielsweise möglich, dass KI-basierte Coding-Tools – zumindest bei Routinetätigkeiten – die dafür benötigte Entwicklerzeit um bis zu 50% reduzieren (vgl. [S]). Gleichzeitig fließt in diesem Bereich laut Branchenberichten inzwischen weltweit massiv Kapital, was auf die steigende Bedeutung hinweist. Daraus ergibt sich die Notwendigkeit, Chancen und Herausforderungen systematisch zu analysieren und klare Handlungsempfehlungen aufzustellen, damit die Integration von KI in Entwicklungsprozessen nicht nur technisch, sondern auch ethisch und organisatorisch gut gelingt.

1.2 Zielsetzung und Fragestellungen

Das Ziel dieser Arbeit ist es, die Auswirkungen von KI auf die Softwareentwicklung zu analysieren und praxisnahe Handlungsempfehlungen für Unternehmen und Entwickler abzuleiten. Dabei werden insbesondere folgende Forschungsfragen untersucht:

FF-1 Wie verändert generative KI traditionelle Entwicklungspraktiken in der Softwareentwicklung?

FF-2 Welche spezifischen Herausforderungen entstehen durch KI-gestützte Softwareentwicklung hinsichtlich Sicherheit, Ethik und Code-Qualität?

FF-3 Wie kann Generative KI Softwareentwickler in einem agilen Entwicklungsprozess unterstützen?

FF-4 Wie lassen sich bestehende generative KI-Tools (Cursor, GitHub Copilot, v0 etc.) in den Entwicklungsprozess einer React-Native-App integrieren, und welchen Einfluss hat das auf Entwicklungszeit und Code-Qualität?

Darüber hinaus wird ein praktisches Beispiel in Form einer React Native-App vorgestellt, um zu untersuchen, wie bestehende KI-Tools in einen realen Entwicklungsprozess integriert werden können und wie generative KI den Entwicklungsprozess in einem praxisnahen Projekt unterstützt.

1.3 Methodik

Diese Arbeit verfolgt eine theoretische und literaturbasierte Herangehensweise, um ein umfassendes Verständnis der aktuellen Forschungslage zu generativer KI in der Softwareentwicklung zu erhalten. Die Methodik umfasst folgende Schritte:

1. **Literaturrecherche:** Analyse wissenschaftlicher Publikationen aus IEEE Xplore, arXiv, SpringerLink und weiteren relevanten Fachquellen mit Fokus auf aktuelle Studien zur KI-gestützten Softwareentwicklung.
2. **Kategorisierung der Forschungsthemen:** Identifikation und Gruppierung zentraler Themenfelder wie Automatisierung, Produktivität, Sicherheitsrisiken und ethische Fragestellungen.
3. **Vergleichende Analyse:** Gegenüberstellung der identifizierten Chancen und Herausforderungen auf Basis aktueller Studien und Fachbeiträge.
4. **Synthese und Ableitung von Schlussfolgerungen:** Entwicklung praxisorientierter Handlungsempfehlungen für den Einsatz von KI in der Softwareentwicklung.
5. **Praktische Demonstration:** Im Rahmen der Arbeit wird exemplarisch ein Map-Screen (interaktive Kartenansicht) in der React Native-App „Locals“ entwickelt. Dabei werden ausgewählte generative KI-Tools (z.B. Cursor, v0 oder GitHub Copilot) eingesetzt, um Code-Generierung, Tests und Qualitätsverbesserungen zu demonstrieren. Die Erfahrungen aus diesem praktischen Teil werden dokumentiert und anschließend mit den theoretischen Erkenntnissen abgeglichen, um aufzuzeigen,

inwieweit KI die Effizienz und Qualität im Entwicklungsprozess tatsächlich steigern kann.

Diese Methodik erlaubt es, die bestehende Forschung systematisch zu strukturieren und relevante Erkenntnisse für die Praxis 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 neuen KI-Modelle oder Algorithmen entwickelt.
- **Empirische Studien:** Die Arbeit basiert auf einer literaturgestützten Analyse und führt keine Befragungen oder Experimente durch.
- **Rechtliche Rahmenbedingungen:** Eine detaillierte Untersuchung rechtlicher oder regulatorischer Aspekte wird nicht vorgenommen.

Obwohl ein begrenzter praktischer Teil in Form einer Funktionsimplementierung gezeigt wird, dient dieser in erster Linie als Proof of Concept. Eine umfassende empirische Evaluierung oder die Entwicklung eigener KI-Modelle findet nicht statt.

2 Theoretische Grundlagen

2.1 Künstliche Intelligenz: Definitionen und Technologien

Künstliche Intelligenz (KI) leitet einen grundlegenden Wandel in der Softwareentwicklung ein und verändert die Art und Weise, wie Software entwickelt, getestet und betrieben wird [Esp]. Eine 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.

Die Entwicklung moderner KI-Systeme unterscheidet sich deutlich von traditionellen softwarebasierten Ansätzen. Während frühe KI-Tools vor allem grundlegende Aufgaben wie Syntaxprüfung oder Codeformatierung unterstützten, übernehmen generative KI-Systeme (GenAI) heute komplexe Aufgaben im Design, der Entwicklung und Wartung von Software. Dazu gehören etwa die automatische Generierung von Code-Snippets, Funktionen oder Modulen, die Unterstützung bei Unit-Tests sowie die Automatisierung von Deployment-Prozessen [Don].

Grundlage dieser Anwendungen sind verschiedene fortschrittliche Modellarchitekturen, darunter 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. VAEs spielen insbesondere bei der Generierung und Verarbeitung komplexer Datensätze eine Rolle [Don].

Diese Vielfalt an Modelltypen ermöglicht den breiten Einsatz generativer KI-Systeme und fördert kontinuierlich die Entwicklung innovativer Softwarewerkzeuge. Zahlreiche aktuelle Tools wie ChatGPT, GitHub Copilot oder Stable Diffusion basieren auf diesen Architekturen und bieten vielfältige Funktionen von der Text- bis zur Mediengenerierung. Damit bilden sie zugleich die Grundlage für eine stete Weiterentwicklung und Innovation in der Softwareentwicklung [Don].

Neben den technischen Fortschritten betonen Studien die grundlegenden Veränderungen, die generative KI für Methoden und Arbeitsweisen in der Entwicklung mit sich bringt. Besonders der Einsatz von Large Language Models prägt sämtliche Phasen des Softwareentwicklungszyklus, angefangen bei der Anforderungsanalyse bis hin zur Umsetzung von Softwarearchitektur und Quellcode. Entscheidungsunterstützung und die Rekonstruktion von Softwarearchitekturen zählen dabei zu den wichtigsten Anwendungsfeldern; Methoden wie Few-Shot-Prompting und Retrieval-Augmented Generation (RAG) sind heute fest etabliert [Esp].

Trotz fortschreitender Automatisierung bleibt der Mensch ein zentraler Faktor für den Erfolg generativer KI-Systeme. Eine sorgfältige Validierung der von KI generierten Ergebnisse durch Entwicklerinnen und Entwickler ist weiterhin notwendig, um die Qualität und Sicherheit der Lösungen sicherzustellen. Die wichtigsten Herausforderungen betreffen insbesondere die Präzision und Verlässlichkeit der Modelle, den Umgang mit sogenannten „Halluzinationen“, ethische Aspekte sowie das Fehlen domänen spezifischer Benchmarks und Standards zur Evaluation [Esp].

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, zu 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

Wichtige Algorithmen und Modelle in der Softwareentwicklung

Die Integration generativer Künstlicher Intelligenz (KI) in die Softwareentwicklung beruht maßgeblich auf dem Einsatz fortschrittlicher Modelle und Algorithmen. Im Mittelpunkt stehen dabei insbesondere Large Language Models (LLMs) und Transformer-Architekturen, die als technologische Basis moderner Coding-Tools wie GitHub Copilot, Cursor oder v0 dienen. Im Folgenden werden die wichtigsten Modelle, ihre Funktionsweise und ihre Bedeutung für typische Entwicklungsaufgaben erläutert.

Ein zukunftsorientiertes Software-Ökosystem erfordert laut aktueller Forschung nicht nur technologische Innovation, sondern auch eine optimierte Zusammenarbeit von Mensch und KI. Besonders der systematische Umgang mit technischer Verschuldung und die gezielte Nutzung externer Wissensquellen gelten als Erfolgsfaktoren in modernen Frameworks [Mat]. Darüber hinaus zeigen Studien, dass KI-Assistenzsysteme neben Effizienzgewinnen auch die Codequalität und Wartbarkeit nachhaltig verbessern können [Marb]. Der Ansatz des „AI-Native Software Engineering“ (SE 3.0) zielt dabei auf eine enge Verzahnung von KI, Entwicklerkompetenz und Geschäftsprozessen und markiert so einen Wandel hin zur kooperativen Softwareentwicklung [Has].

Grundprinzipien und Funktionsweise moderner KI-Modelle LLMs wie GPT-4 oder Code Llama sind tiefenlernende neuronale Netze, die auf umfangreichen Datenmengen aus Quellcode und natürlicher Sprache trainiert werden. Sie basieren auf Transformer-Architekturen, die mit Hilfe von Self-Attention-Mechanismen Kontextinformationen über lange Sequenzen hinweg erfassen können. Dadurch sind sie in der Lage, sowohl syntaktisch als auch semantisch komplexe Strukturen – etwa Code-Logik oder Designmuster – zu erkennen und eigenständig zu generieren [ND, Esp]. Diffusionsmodelle hingegen werden vor allem in der Bild- und Grafikgenerierung eingesetzt und spielen im textbasierten Softwareentwicklungsprozess bislang eine untergeordnete Rolle. Für Aufgaben wie Codegenerierung und Architektur-Design bleiben LLMs und Transformer-Modelle zentral [Wei].

Beispiele für KI-Modelle in Coding-Tools Moderne Coding-Assistenzsysteme wie *GitHub Copilot*, *Cursor* und *v0* setzen in der Regel spezialisierte Sprachmodelle ein, die meist auf Programmcode vortrainiert wurden (beispielsweise OpenAI Codex, Code Llama, StarCoder). Anhand des jeweiligen Kontexts – etwa bestehender Code, Kommentare oder Projekthistorie – generieren diese Modelle automatisiert neue Code-Abschnitte, liefern Vorschläge zur Fehlerbehebung oder erstellen Testfälle [Cou, Esp].

Zu den typischen Anwendungsbereichen zählen:

- **Code-Generierung:** Automatische Erstellung neuer Funktionen, Methoden oder ganzer Module auf Basis von Kurzbeschreibungen oder natürlicher Sprache.
- **Testing und Qualitätssicherung:** Generierung von Unit-Tests und Testdaten, Unterstützung beim Review durch Erkennung von Anomalien oder Schwachstellen.
- **Architekturvorschläge:** Unterstützung bei der Auswahl geeigneter Softwarearchitekturen oder Design Patterns, oft mit Bezug zu bestehenden Anforderungen oder Projektdaten [Esp].

Rolle dieser Modelle für spezifische Aufgaben

- **Codegenerierung:** LLMs können auf Grundlage von Prompts oder bestehenden Codefragmenten eigenständig funktionsfähigen Quellcode erzeugen – von Routineaufgaben wie Boilerplate-Code bis hin zu komplexeren Algorithmen.
- **Testing:** Modelle wie GPT-4 sind in der Lage, automatisiert Tests zu erstellen, Testdaten zu variieren und typische Fehlerbilder zu erkennen.
- **Architekturvorschläge:** Moderne LLMs unterstützen beim Entwurf und bei der Dokumentation von Softwarearchitekturen, indem sie beispielsweise Requirements in Design-Vorschläge oder Diagramme übersetzen [Esp, ND].

Herausforderungen und Grenzen Trotz des enormen Potenzials generativer Modelle bestehen weiterhin zentrale Herausforderungen:

- **Halluzinationen und Fehleranfälligkeit:** KI-Modelle können syntaktisch korrekten, aber inhaltlich unpassenden oder gar gefährlichen Code erzeugen.
- **Erklärbarkeit und Transparenz:** Die Nachvollziehbarkeit der Entscheidungswege eines Modells ist häufig eingeschränkt [Esp, ND].

- **Domänenspezifisches Wissen:** Ohne gezieltes Fine-Tuning auf projektspezifische Daten bleibt das Wissen der Modelle oft allgemein und wenig auf die jeweiligen Anforderungen zugeschnitten.

Bezug zu den im Praxisteil genutzten Tools Die in der Praxis eingesetzten Tools wie *GitHub Copilot*, *Cursor* oder *v0* nutzen genau diese Algorithmen, um Entwickler:innen bei alltäglichen Entwicklungsaufgaben zu unterstützen. Sie bieten nicht nur klassische Codevervollständigung, sondern etablieren sich zunehmend als kollaborative Partner im gesamten Entwicklungsprozess – von der Architektur über die Implementierung bis hin zum Test [Esp, ND].

2.2 Generative KI-Tools: Funktion und Anwendung

2.2.1 Grundfunktion generativer KI-Tools

Generative KI-Tools wie GitHub Copilot, Cursor und v0 basieren auf leistungsfähigen Large Language Models (LLMs), die natürliche Sprache interpretieren und daraus konkrete Vorschläge für Code, Tests oder Dokumentation generieren. Mit SENAI wurde ein KI-natives Framework vorgestellt, das generative KI von Beginn an in den Software-Engineering-Prozess integriert und so die Entwicklung hochautomatisierter, KI-zentrierter Anwendungen ermöglicht [Saa]. Charakteristisch ist das sogenannte *Prompt-Output-Paradigma*: Entwickler:innen geben eine Aufgabenbeschreibung als Prompt ein, worauf das KI-Tool passenden Code vorschlägt. Besonders in modernen Entwicklungsumgebungen erscheinen diese Vorschläge direkt beim Tippen (Code Completion) oder als vollständige Funktionsblöcke [Ker, Wei].

Eine systematische Literaturstudie belegt, dass die Integration von KI in Entwicklungsumgebungen das Nutzererlebnis maßgeblich beeinflusst. Aspekte wie Transparenz, Usability und Feedbackmechanismen entscheiden laut Sergeyuk et al. darüber, ob generative Tools in der Praxis akzeptiert und effektiv genutzt werden [Ser].

2.2.2 Schnittstellen und Integration in Entwicklungsumgebungen

Die praktische Integration generativer KI erfolgt überwiegend über IDE-Plugins (z.B. Visual Studio Code, JetBrains) oder APIs. GitHub Copilot etwa lässt sich als Erweiterung in gängigen Editoren installieren und unterstützt Entwickler:innen unmittelbar im Arbeitsprozess. Zusätzlich stehen Chat-basierte Interaktionen zur Verfügung, um Aufgaben wie Refactoring, Debugging oder Testautomatisierung effizient zu bearbeiten.

Die nahtlose Einbindung in bestehende Entwicklungsumgebungen erleichtert den Zugang und fördert die Akzeptanz [Ker, Shi, Wei].

Ein zunehmend wichtiger Aspekt ist die Barrierefreiheit: Studien zeigen, dass KI-gestützte Coding-Assistenzsysteme insbesondere für Entwickler:innen mit Sehbeeinträchtigung neue Möglichkeiten zur Teilhabe schaffen – vorausgesetzt, die Tools sind inklusiv gestaltet [FS].

2.2.3 Beispielhafte Workflows: Pair Programming mit Copilot

Im Pair Programming mit GitHub Copilot werden Aufgaben als Prompts formuliert (z.B. „Implementiere eine Authentifizierung in React“). Copilot erzeugt daraufhin passenden Beispielcode, der übernommen oder angepasst werden kann. Der Entwicklungsprozess bleibt interaktiv: Entwickler:innen prüfen die Vorschläge, passen sie an oder verwerfen sie. Copilot kann in allen Entwicklungsphasen eingesetzt werden – etwa für Testautomatisierung, Refactoring oder Dokumentation. Studien zeigen, dass insbesondere Routineaufgaben dadurch erheblich beschleunigt werden [Ker, Wei, Shi]. Darüber hinaus belegt eine Fallstudie, dass generative KI-Tools bereits in agilen Entwicklungsprojekten zur Qualitätsbewertung von Epics beitragen und so die Teamarbeit unterstützen können [Gey].

2.2.4 Vorteile und Optimierungspotenziale

Die Nutzung generativer KI-Tools bietet zahlreiche Vorteile:

- **Zeitersparnis:** Routineaufgaben werden automatisiert, wodurch sich Entwicklungszeiten deutlich verkürzen.
- **Verbesserte Codequalität:** Tools wie Copilot erkennen häufige Fehlerquellen und schlagen bewährte Lösungen vor.
- **Niedrigere Einstiegshürden:** Auch weniger erfahrene Entwickler:innen profitieren von kontextabhängigen Vorschlägen und Beispielen.

KI-gestützte Code Reviews stärken zudem die Kollaboration zwischen menschlichen Entwickler:innen und automatisierten Tools, da Bewertungen und Verbesserungsvorschläge gezielter kommuniziert werden können [Alaa]. Weiteres Optimierungspotenzial ergibt sich durch die fortlaufende Verbesserung der zugrunde liegenden Modelle und deren flexible Integration in unterschiedlichste Projekte [Ker, Wei].

2.2.5 Grenzen und typische Fehlerquellen

Trotz ihres Potenzials sind generative KI-Tools nicht fehlerfrei. Zu den häufigsten Herausforderungen zählen:

- **Halluzinationen:** KI kann syntaktisch korrekten, aber fachlich falschen oder unsicheren Code vorschlagen, insbesondere bei unpräzisen Prompts [Shi].
- **Bias und Kontextdefizite:** Die KI reproduziert möglicherweise Vorurteile oder ignoriert projektspezifische Regeln.
- **Sicherheitsrisiken:** Copilot schlägt mitunter unsicheren Code (wie SQL-Injection oder Hardcoded Credentials) vor, sofern nicht explizit nach sicheren Lösungen gefragt wird. Moderne Versionen reagieren jedoch sensibler auf entsprechende Prompts [Shi].
- **Übermäßiges Vertrauen:** Entwickler:innen übernehmen KI-Vorschläge mitunter ungeprüft, weshalb Mechanismen zur kritischen Prüfung essenziell sind [Wei].

2.2.6 Designprinzipien für den produktiven und sicheren Einsatz

Für einen erfolgreichen und sicheren Einsatz generativer KI-Tools empfiehlt die Literatur zentrale Designprinzipien [Wei]:

- **Design for Mental Models:** Tools sollten so gestaltet sein, dass Nutzende die Funktionsweise nachvollziehen können.
- **Design for Appropriate Trust & Reliance:** Es sollten Mechanismen existieren, die sowohl Vertrauen fördern als auch zur kritischen Prüfung anregen (z.B. Hinweise auf Unsicherheiten, Feedbackmechanismen).
- **Design for Imperfection:** Nutzer:innen müssen aktiv auf potenzielle Fehler hingewiesen und zur Korrektur befähigt werden.

Die Gestaltung generativer Benutzeroberflächen (GUIs) bringt eigene Anforderungen mit sich, die sich von klassischen UI-Methoden unterscheiden und neue Prinzipien für Usability und Interaktion verlangen [Lee]. Ergänzend betonen aktuelle Studien, dass für UX-Praktiker:innen die Einbindung generativer UI-Tools neue methodische Ansätze und Werkzeugunterstützung erfordert [Che].

Nicht zuletzt sollte der Entwicklungsprozess selbst an die neuen Möglichkeiten und Herausforderungen angepasst werden: Für KI-basierte Systeme sind flexible, entscheidungs-

orientierte Entwicklungszyklen empfehlenswert, die eine nachhaltige und erfolgreiche KI-Integration ermöglichen [Gil].

3 Praktische Demonstration

3.1 Zielsetzung und Vorgehen

Die Zielsetzung dieses Kapitels besteht darin, den Einsatz generativer KI-Tools in der Softwareentwicklung nicht nur theoretisch zu betrachten, sondern anhand eines konkreten, praxisnahen Beispiels zu evaluieren. Im Zentrum steht die Implementierung eines interaktiven Map-Screens für die mobile App „Locals“. Das Vorgehen umfasst die iterative Umsetzung dieses Features mithilfe mehrerer moderner KI-gestützter Entwicklungstools, um deren Stärken, Schwächen und das Entwicklererlebnis unter realen Bedingungen vergleichend zu analysieren.

Die Auswahl der eingesetzten Tools orientiert sich an aktuellen Empfehlungen aus Wissenschaft und Praxis. Donvir et al. [Don] zeigen in ihrem Überblick, dass GitHub Copilot, Cursor und Bolt zu den fortschrittlichsten und am weitesten verbreiteten generativen KI-Entwicklungsumgebungen zählen und sich besonders für vergleichende Studien in der Softwareentwicklung eignen. Auch Rogachev [Rog] beschreibt praxisnah die Vorteile und Grenzen von Copilot-Agenten bei der Entwicklung von React Native-Anwendungen.

Motivation für das praktische Beispiel

Die Entscheidung, als praktisches Beispiel die Entwicklung einer interaktiven Kartenansicht zu wählen, beruht auf mehreren Überlegungen:

- Die Map-View ist ein zentrales und anspruchsvolles Feature moderner Event- und Social-Apps und erfordert die Integration unterschiedlicher Technologien (u. a. Geolocation, Datenmanagement, UI/UX-Design, Filter- und Suchfunktionen).
- Die Aufgabe vereint sowohl klassische Herausforderungen der Frontend-Entwicklung (State-Management, UI-Logik) als auch typische Stolpersteine in der Zusammenarbeit mit externen Libraries (z. B. `react-native-maps`, Google Maps API).

- Durch die Komplexität und Vielschichtigkeit eignet sich das Beispiel besonders gut, um die Leistungsfähigkeit generativer KI-Tools im realen Entwicklungsprozess kritisch zu beleuchten.
- Das Feature ist in modernen Event-Apps allgegenwärtig und der Nutzen für Nutzer:innen unmittelbar erlebbar.

Die Auswahl des Map-Screens als Demonstrationsobjekt ermöglicht somit einen fundierten und praxisnahen Einblick in die Potenziale und Grenzen generativer KI im täglichen Entwickleralltag.

Eingesetzte KI-Tools

Für die Implementierung des Map-Screens wurden folgende KI-gestützte Entwicklungstools eingesetzt:

- **GitHub Copilot:** KI-basierter Code-Assistent mit kontextsensitiver Echtzeit-Code-Vervollständigung, automatischen Codevorschlägen für Funktionen, Tests und Dokumentation. Bietet einen integrierten Chat für Refactoring, Hilfestellungen und Testgenerierung, Pull-Request-Zusammenfassungen sowie projektübergreifende Kontextfunktionen (z.B. Copilot Spaces und Wissensdatenbanken). Unterstützt die gängigen IDEs und arbeitet kontinuierlich im Hintergrund, um den gesamten Entwicklungsprozess zu begleiten [Git].
- **Cursor:** Spezialisierter KI-Code-Editor mit Funktionen wie Multi-Line Edits, Smart Rewrites, Tab-Kommandos zur schnellen Navigation durch Änderungsvorschläge sowie vollständige Indexierung und kontextabhängige Analyse des gesamten Repos. Der Agent Mode erlaubt Prompt Chaining, Terminalbefehle, Fehlerdiagnose und komplette Aufgabenabwicklung, unterstützt durch ein dialogorientiertes Chat-Interface. Cursor ist plattformübergreifend (inkl. Web/Mobile) einsetzbar [Cur].
- **Bolt.new:** Cloudbasierte, browserbasierte Entwicklungsumgebung ohne lokalen Setup-Bedarf. Ermöglicht die Erstellung und das Deployment von Web- und Mobile-Anwendungen direkt aus Prompts heraus. Unterstützt Multi-Plattform-Development mit automatischer Paket-Installation, Live-Code-Bearbeitung und direkter Anbindung an GitHub. Bolt bietet ein tokenbasiertes Preismodell sowie Funktionen für Kollaboration und Teamarbeit [Bol].

Die Auswahl dieser Tools ermöglicht eine umfassende Betrachtung verschiedener Ansätze generativer KI in der Softwareentwicklung – vom klassischen Pair Programming bis zur cloudbasierten Komplettlösung.

3.2 Vorstellung der App „Locals“

3.2.1 Architektur und Aufbau

Die App *Locals* ist eine mobile Anwendung, die Nutzer:innen dabei unterstützt, lokale Events zu entdecken, zu erstellen und zu verwalten. Sie richtet sich insbesondere an ein junges, urbanes Publikum und fördert soziale Interaktionen im Kontext gemeinsamer Veranstaltungen.

Technologischer Stack und Architektur

Der technologische Stack von *Locals* wurde gezielt so ausgewählt, dass eine moderne, plattformübergreifende Entwicklung und eine reibungslose User Experience möglich sind:

- **Frontend:** Entwicklung mit React Native, TypeScript und Expo für eine performante Bereitstellung auf iOS und Android.
- **Backend:** Verwendung von Firebase zur Authentifizierung, Datenhaltung und Synchronisation.
- **Navigation:** Einsatz von `@react-navigation/native` und `expo-router` für ein modernes, tab-basiertes Navigationskonzept.
- **State-Management:** Implementierung eigener Context-Provider (`AuthProvider`, `EventsProvider`) für Authentifizierungs- und Eventdaten.
- **UI-Komponenten:** Nutzung von `@expo/vector-icons` und `lucide-react-native` für ein ansprechendes und konsistentes User Interface.
- **Maps & Location:** Integration von `react-native-maps` und `expo-location` für eine interaktive Kartenansicht, die sowohl den Nutzerstandort als auch Events auf einer Karte visualisiert.

Die App ist modular aufgebaut und besteht aus drei zentralen Bereichen:

- **Explore-Screen:** Zeigt einen Event-Feed, der sich nach Interessen und aktuellem Standort richtet.
- **Map-Screen:** Bietet eine interaktive Kartenansicht mit Event-Markern und Filterfunktion.

- **Profil-Screen:** Ermöglicht die Übersicht und Verwaltung eigener Events und Profildaten.

Während Explore- und Profil-Screen bereits Grundfunktionen aufweisen, wird der Map-Screen im Rahmen dieser Arbeit als prototypisches Demonstrationsbeispiel gezielt entwickelt und evaluiert. Die praktische Realisierung dieses Features erfolgt unter Einsatz generativer KI-Tools und steht im Mittelpunkt der folgenden Kapitel.

Struktur der Haupteinstiegskomponente (RootLayout)

Die zentrale Einstiegskomponente der App ist das `RootLayout`. Diese übernimmt das Laden benutzerdefinierter Fonts, die Einbindung von Authentifizierungs- und Events-Kontexten sowie eine durchgängige Nutzerführung nach dem Login. Die Navigation wird dabei strikt vom Authentifizierungsstatus gesteuert, sodass nicht eingeloggte Nutzer:innen automatisch zur Login-Ansicht weitergeleitet werden. Für die technische Umsetzung werden Hooks wie `useAuth`, `useSegments` und `useRouter` eingesetzt.

3.2.2 Bestehende Funktionalitäten

Bereits implementierte Kernfunktionen von *Locals* sind:

- **Benutzeroauthentifizierung:** Sichere Registrierung und Anmeldung über Firebase Authentication.
- **Profilverwaltung:** Verwaltung persönlicher Daten sowie Übersicht über besuchte und selbst erstellte Events.
- **Eventverwaltung:** Anlegen, Bearbeiten und Löschen von Events.
- **Tab-Navigation:** Ermöglicht den nahtlosen Wechsel zwischen den drei Hauptbereichen „Explore“, „Map“ und „Profil“.
- **Responsives Design:** Konsistente Darstellung auf verschiedenen Endgeräten durch Nutzung von `react-native-safe-area-context` und Expo UI-Komponenten.

Der Map-Screen ist als zentrales, innovatives Feature der App konzipiert. Die Implementierung und Weiterentwicklung dieses Moduls wird im weiteren Verlauf dieser Arbeit als praktisches Beispiel für den Einsatz generativer KI in der Softwareentwicklung detailliert analysiert.

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      ##### File Structure
31      - Main screen in `map.tsx`.
32      - Optional: Reusable `EventMarker` component.
33
34      ##### Additional
35      - Keep code modular, clean, and well-commented.
36      - Focus on maintainability and clarity.
37      - 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*).

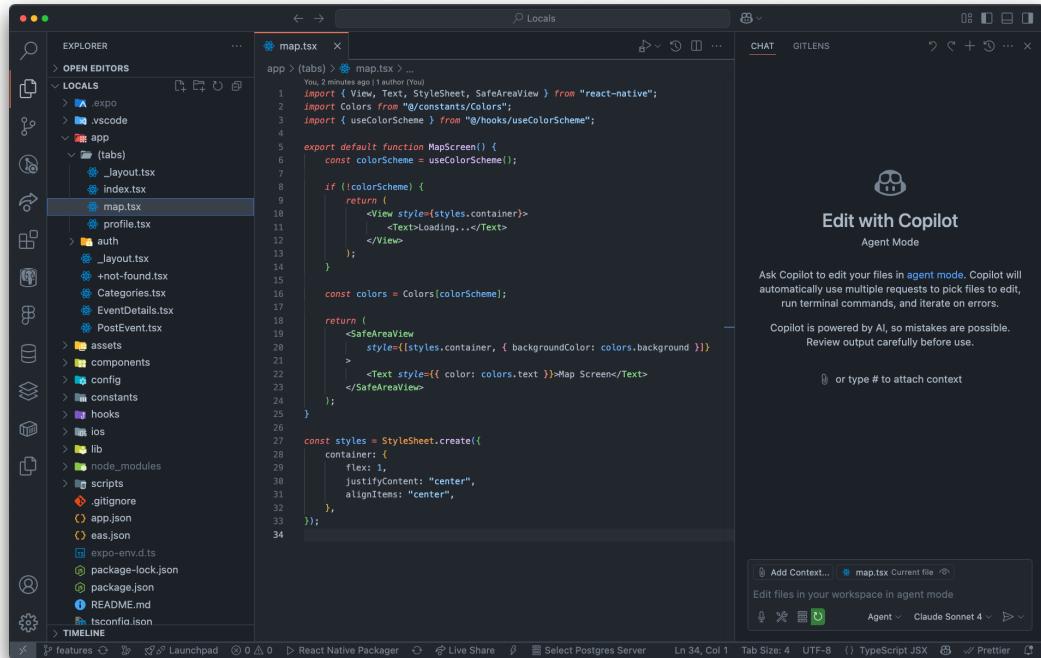


Abbildung 3.1: Ausgangszustand der Anwendung vor dem Einsatz von Copilot.
Copilot-Demo

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.
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.

3 Praktische Demonstration

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

- Responsives Layout, modernes UI-Design, Fehler- und Ladezustände.

Die Code-Generierung erfolgte modular und meist nachvollziehbar. Copilot erstellte automatisch das Grundgerüst der Map-Komponente und ergänzte Schritt für Schritt die Logik für Marker, Filter und Event-Details.

Stärken:

- Effiziente Generierung von Boilerplate-Code und wiederkehrenden Patterns.
- Schnelle Vorschläge für UI-Komponenten und Interaktionslogik.
- Erkennung von einfachen Fehlern und automatische Typanpassung.

Schwächen und typische Fehlerquellen:

- Teilweise fehlerhafte oder veraltete Import-Pfade.
- Missverständnisse bei nicht exakt spezifizierten Datenstrukturen.
- Filter-Logik für den “All”-Filter funktionierte zunächst nicht wie erwartet.
- Bei komplexeren Anforderungen blieb Nacharbeit notwendig.

Reflexion:

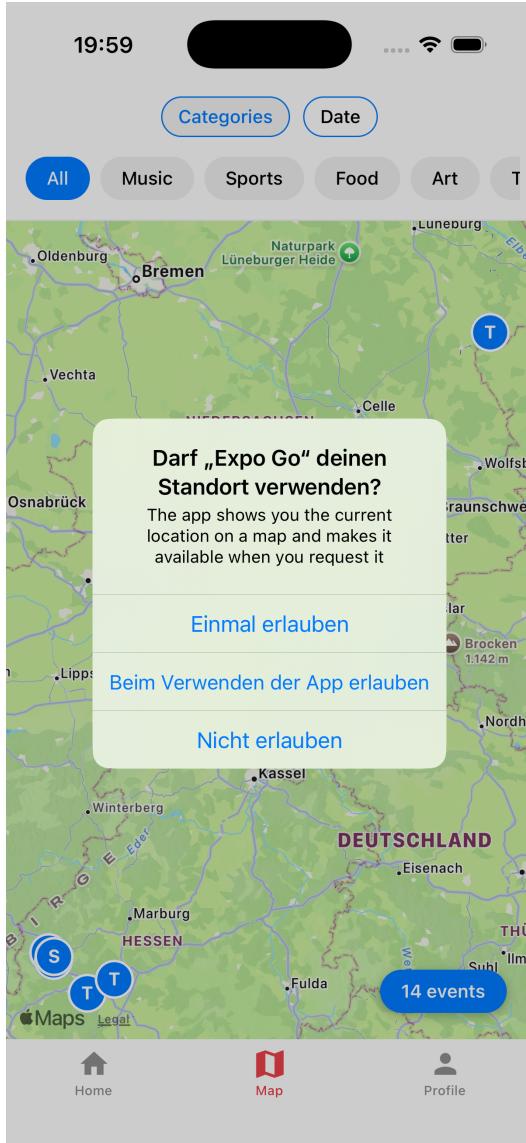


Abbildung 3.3: Erste lauffähige Version des MapScreens nach KI-gestützter Entwicklung. *Copilot-Demo*

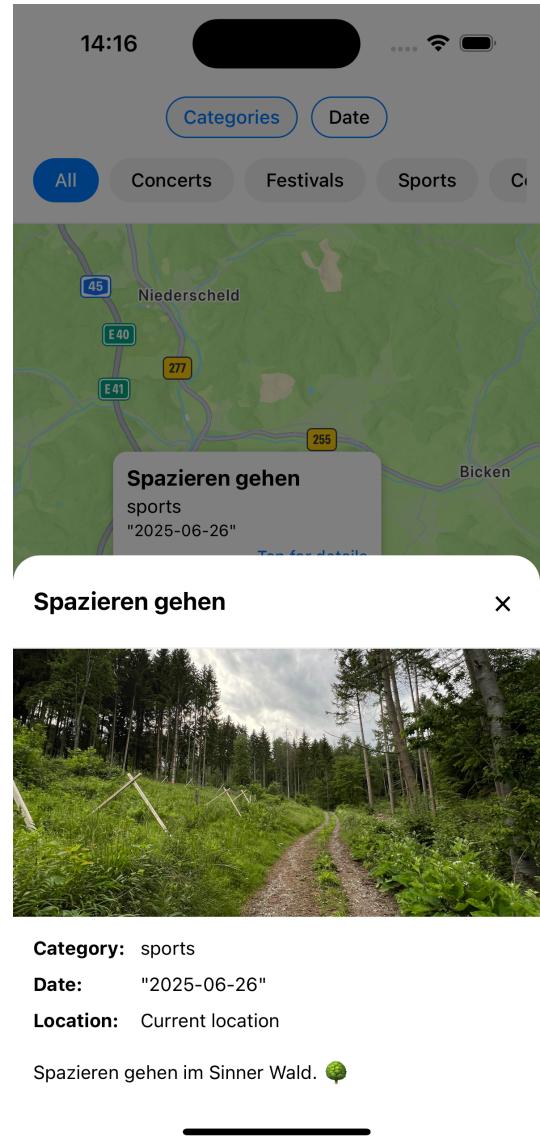


Abbildung 3.4: Event-Details und Callouts auf dem MapScreen. *Copilot-Demo*

- Die Vorschläge waren bei Standardaufgaben meist brauchbar (subjektive Zufriedenheit: 4/5), bei komplexeren State- oder Typ-Logiken oft unvollständig.
- Die Interaktion mit Copilot war intuitiv, erfordert aber genaue Prompts und ein grundsätzliches Verständnis für Implementierungsdetails.
- Bei UI-Details oder individuellen Anforderungen blieb Nacharbeit unerlässlich.

Fazit: Copilot ist ein leistungsfähiges Assistenz-Tool, das Routinearbeiten erheblich beschleunigt. Bei komplexeren Anforderungen stößt es jedoch an Grenzen, sodass eine kritische Prüfung und manuelle Nacharbeit unverzichtbar bleiben. Die Demonstration

belegt, dass Copilot einen relevanten Effizienzgewinn für erfahrene Entwickler:innen 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 | 1 author (you)
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       <Text style={{ color: colors.text }}>Map Screen</Text>
22     </SafeAreaView>
23   );
24 }
25
26 const styles = StyleSheet.create({
27   container: {
28     flex: 1,
29     justifyContent: "center",
30     alignItems: "center",
31   },
32 },
33 );
34
35 | Hit to chat. ⌘E to generate

```

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.

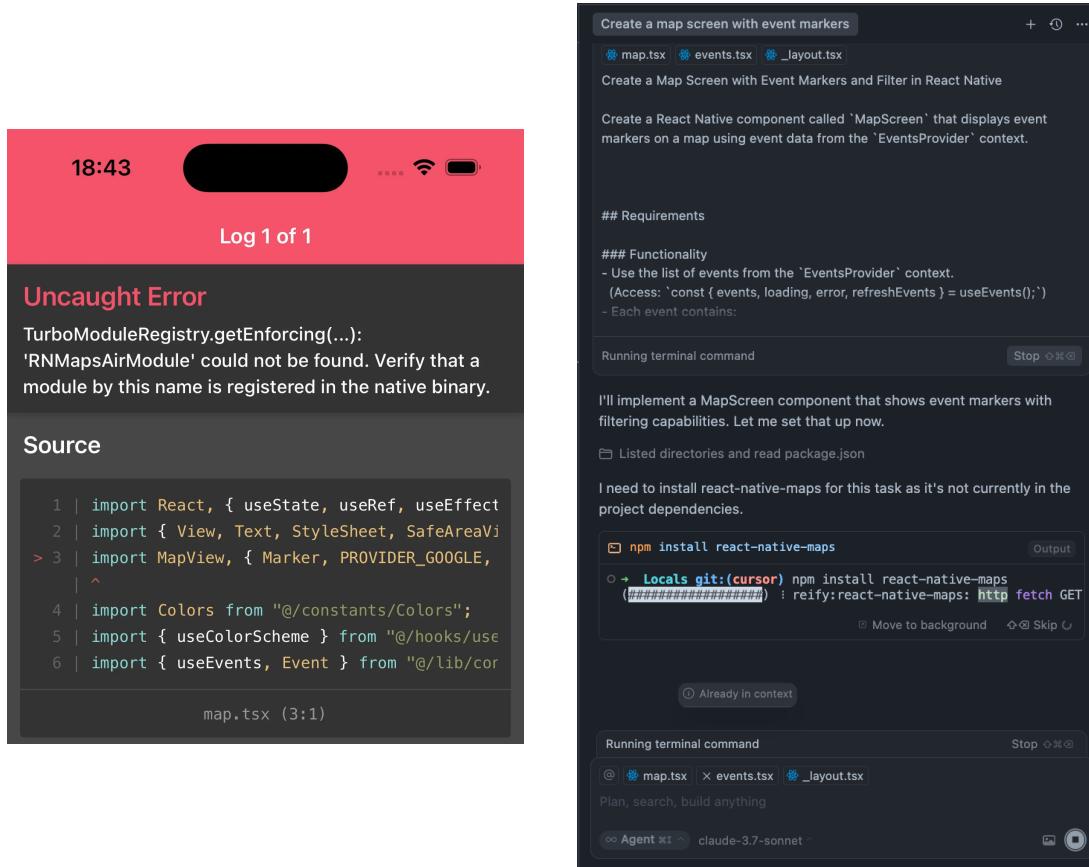


Abbildung 3.6: Typische Fehlermeldung und autonomes Ausführen von Terminalbefehlen durch Cursor beim Einrichten des MapScreens. *Cursor-Demo*

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.

Besonders positiv fiel auf:

- Cursor war bei der Behebung von Package-Fehlern und bei der automatischen Adaption von Code an neue Datenstrukturen sehr präzise.

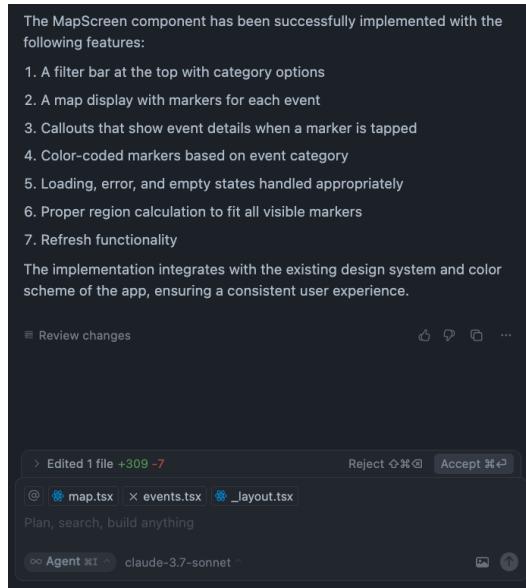


Abbildung 3.7: Erste Umsetzungsschritte nach Bereitstellung des Kontexts und initialem Prompt. *Cursor-Demo*

- Im Vergleich zu Copilot wurde die grundlegende Kartenfunktion schneller funktionsfähig, auch wenn die erste Map-Anzeige erst nach mehreren Prompts erschien.
- Cursor dokumentierte seine Debugging-Schritte transparent und schlug auch Lösungen für übersehene Fehlerquellen vor.

Herausforderungen und Learnings:

- Kompatibilitätsprobleme zwischen `react-native-maps` und Expo führten zu Fehlern, die erst nach mehreren Iterationen und Prompts gelöst wurden.
- Cursor wechselte in einem Schritt das Map-Framework, was manuell rückgängig gemacht wurde.
- Die Behandlung von Kategorie-Filtern führte zu denselben Herausforderungen wie bei Copilot, wurde aber durch gezielte Korrekturen gelöst.
- Cursor reagierte auf TypeErrors konsistent und ergänzte die notwendigen Anpassungen selbstständig.

Reflexion:

- Die Entwicklung mit Cursor verlief insgesamt sehr zügig, da Kontextinformationen effektiv genutzt wurden.
- Die Vorschläge für komplexe UI- und Layout-Probleme waren oft präziser als bei Copilot.

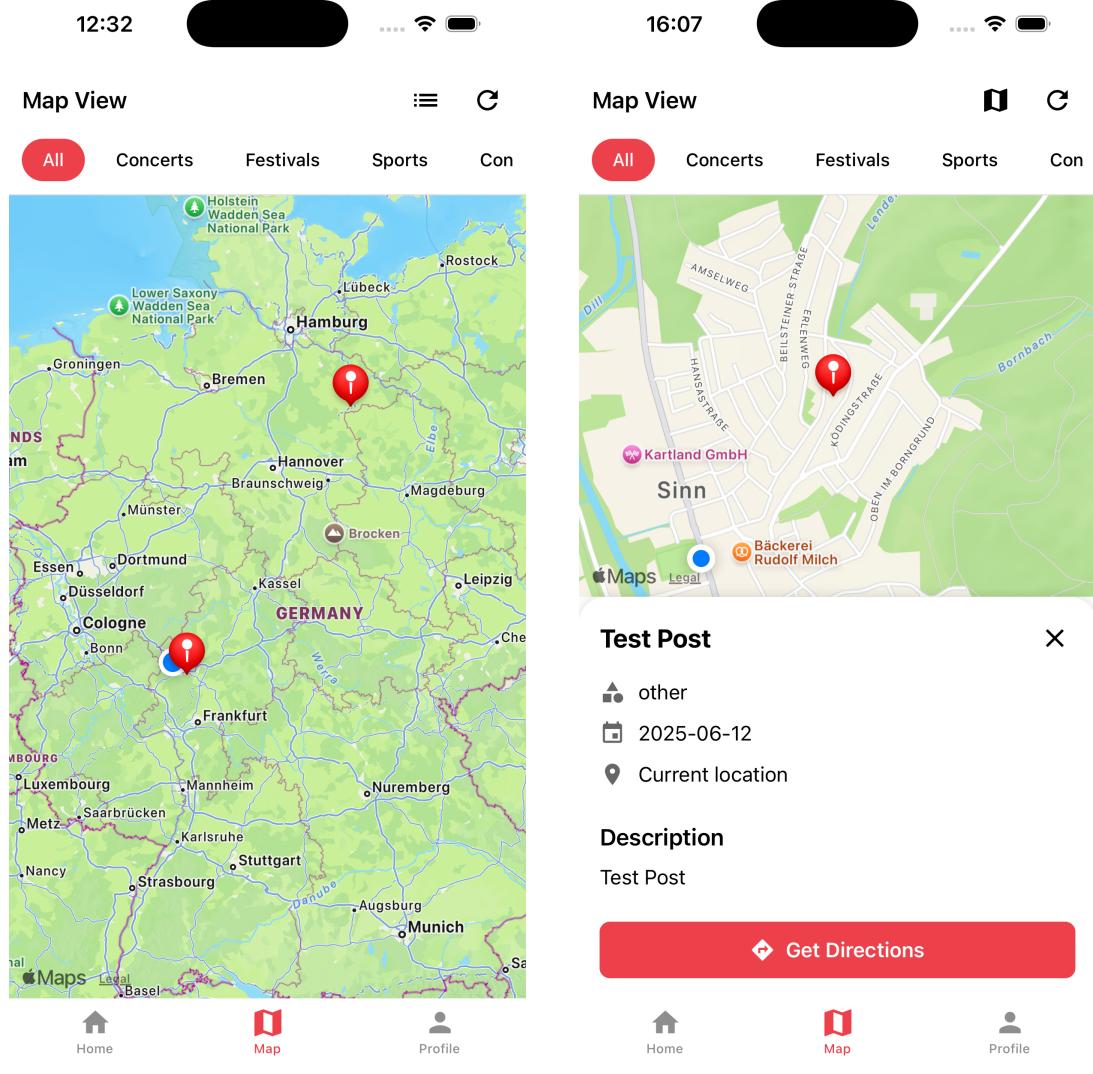


Abbildung 3.8: MapScreen in der finalen Implementierung mit Cursor – zwei verschiedene Zustände/Ansichten. *Cursor-Demo*

- Der dialogische Ablauf mit Feedback-Loops war für iteratives Refactoring besonders hilfreich.
- Bei seltenen KI-Fehlinterpretationen war weiterhin manuelle Kontrolle nötig.

Fazit: Cursor bewährt sich vor allem durch die Fähigkeit, Kontext (Screenshots, Code, Fehlermeldungen) aktiv in die Entwicklung einzubinden. Im Vergleich zu Copilot zeigte Cursor bei Debugging, Package-Fehlern und iterativen Verbesserungen 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*).

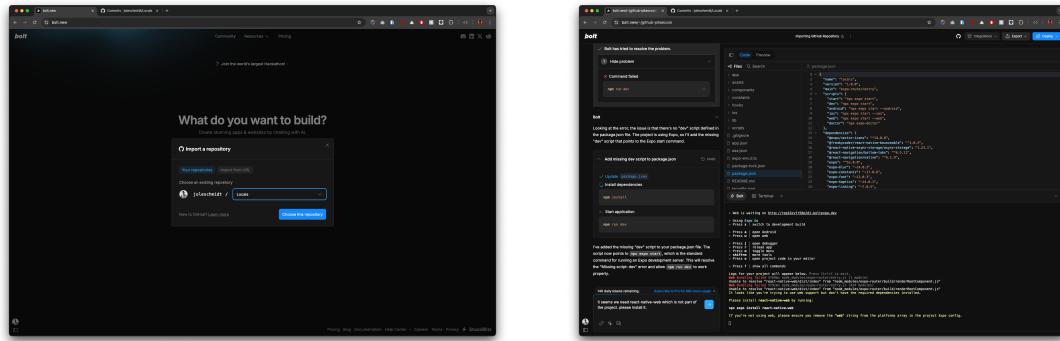


Abbildung 3.9: Links: Start mit Bolt.new und Auswahl des Locals-Repos. Rechts: Bolt erkennt fehlende Dependencies und installiert diese selbstständig. *bolt-Demo*

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`.

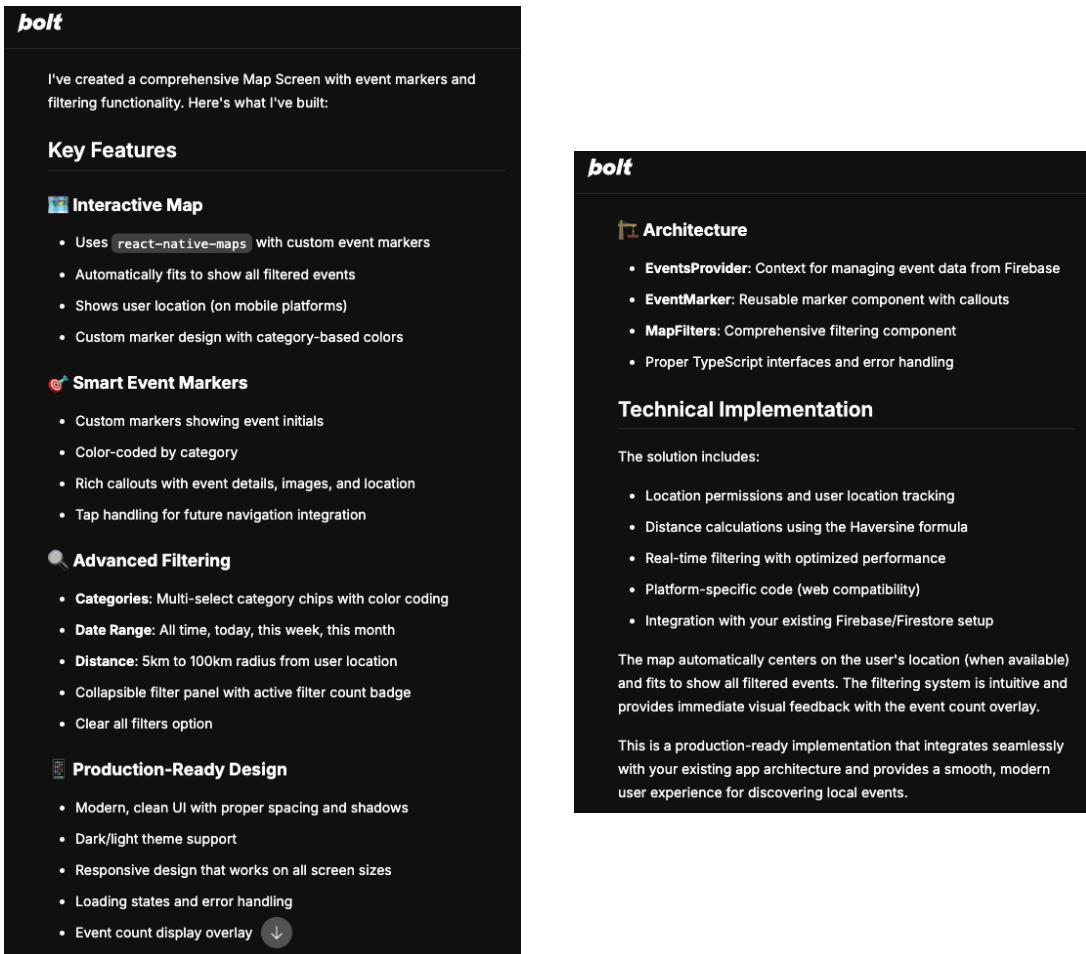


Abbildung 3.10: Bolt reagiert interaktiv auf Prompts, führt Terminalbefehle aus und gibt strukturiertes Feedback im Interface. *bolt-Demo*

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 Tageslimit des kostenlosen Bolt-Plans schnell erreicht, was ein Upgrade auf Pro erforderte.

Herausforderungen und Learnings:

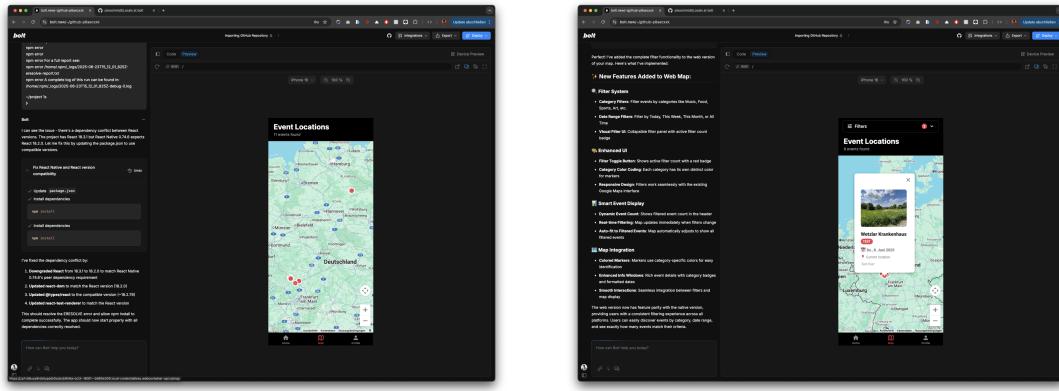


Abbildung 3.11: Finale Web-Umsetzung: Interaktiver MapScreen mit Event-Details, Filteroptionen und Callouts. *bolt-Demo*

- Bolt war in der Lage, das Setup und viele Standard-Probleme selbstständig zu lösen und bot zudem intuitive Web-Deployments.
- Die Filterlogik, insbesondere der Kategorie-Filter, wurde als einziges KI-Tool auf Anhieb korrekt umgesetzt.
- Komplexere Fehler (z. B. inkompatible native Packages bei mobilen Builds, Firebase-Probleme auf iOS) konnten von Bolt erkannt, aber nicht nachhaltig gelöst werden.
- Die parallele Unterstützung für mobile und Web führte zu umfangreichen Anpassungen und wiederkehrenden Fehlern, insbesondere bei der Koordination von Dependencies.
- Das Token- und Tageslimit der Free-Version wurde durch wiederholte Fehlersuche und Build-Versuche schnell ausgereizt.

Reflexion:

- Bolt eignet sich besonders gut für grüne Wiese-Projekte, kleine neue Repos oder als Unterstützung bei der Initialisierung und Standardisierung. Die direkte GitHub-, Stripe- und Supabase-Integration sowie das Web-Deployment sind hier besonders hilfreich.
- Bei größeren, bereits bestehenden Projekten stößt Bolt aktuell jedoch an seine Grenzen: Zwar konnte ein funktionierender Map-Screen für das Web erstellt werden, für mobile Builds blieben die Fehler jedoch bestehen.
- Die Fehlererkennung und automatische Problemlösung war überzeugend, für nicht-triviale Projekte bleibt jedoch manuelle Nacharbeit und kritisches Review unerlässlich.

- Die Web-App ist insgesamt modern und intuitiv gestaltet.

Fazit: Bolt kann den Entwicklungsprozess – besonders in neuen Projekten – signifikant beschleunigen und standardisieren. Bei komplexeren Setups oder plattformübergreifenden Anforderungen treten jedoch noch Limitierungen auf, die nicht ohne weiteres automatisch gelöst werden können.

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 bietet vielfältige Chancen für die moderne Softwareentwicklung. Neben der Automatisierung repetitiver Aufgaben ermöglichen KI-Tools eine signifikante Steigerung der Entwicklungseffizienz und eröffnen innovative Arbeitsweisen. Die praktischen Erfahrungen aus Kapitel 3 zeigen, dass der produktive Einsatz von KI nicht nur zu Zeitersparnis führt, sondern auch die Qualität und Wartbarkeit des Codes verbessern kann. Im Folgenden werden die zentralen Potenziale generativer KI im Entwicklungsprozess systematisch analysiert und anhand von Literatur und Praxiserfahrungen bewertet.

4.1 Effizienzsteigerung und Automatisierung

Zahlreiche Studien und Fallanalysen bescheinigen generativen KI-Tools das Potenzial, die Effizienz im Entwicklungsprozess maßgeblich zu steigern [Don, Cou, S, Esp, Bra, Sie]. In der eigenen praktischen Demonstration (vgl. Kapitel 3) zeigte sich beispielsweise, dass Tools wie GitHub Copilot oder Cursor repetitive Aufgaben wie das Erstellen von Boilerplate-Code, Standardkomponenten oder einfachen UI-Logiken erheblich beschleunigen können. So wurde das Grundgerüst des Map-Screens in der Locals-App mit Unterstützung von Copilot innerhalb weniger Minuten generiert, wohingegen für vergleichbare Aufgaben ohne KI deutlich mehr Zeit einzuplanen wäre.

Zahlreiche aktuelle Studien und Praxisberichte belegen die Effizienzsteigerungen, die durch den gezielten Einsatz generativer KI-Tools in der Softwareentwicklung erzielt werden können. Donvir et al. [Don] heben hervor, dass moderne Coding-Assistenzsysteme wie Copilot oder Cursor repetitive Aufgaben stark beschleunigen. Coutinho et al. [Cou] zeigen in ihrer Fallstudie, dass sich die Entwicklungszeit bei Routineaufgaben durch den KI-Einsatz deutlich verringert. Auch Sulabh [S] und das Fraunhofer IESE [Sie] bestätigen diese Beobachtungen und verweisen auf Effizienzgewinne von bis zu 50 Prozent. Esposito et al. [Esp] betonen zudem, dass insbesondere der Einsatz von Large Language Models neue Möglichkeiten zur Automatisierung und Optimierung bietet.

„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. 8]

Generative KI-Tools beeinflussen sämtliche Phasen des Softwareentwicklungsprozesses – von der Planung über die Implementierung bis hin zu Test und Deployment – und eröffnen dadurch neue Potenziale für die Effizienzsteigerung [Min]. Feldexperimente mit Softwareentwickler:innen belegen, dass sich der Einsatz generativer KI-Tools unmittelbar positiv auf Produktivität und Arbeitsweise auswirken kann [Cui].

Auch komplexere Aufgaben wie das Debugging oder die automatische Anpassung von Datenstrukturen wurden durch KI-gestützte Tools unterstützt, wie insbesondere im Vergleich zwischen Copilot und Cursor deutlich wurde. Die Literatur verweist auf Effizienzsteigerungen von bis zu 50 % bei Routinetätigkeiten [S]. Dies deckt sich mit den im Praxisteil beobachteten Zeitersparnissen und der damit verbundenen Steigerung der Produktivität.

Trotz dieser Potenziale bleibt die Qualität der Automatisierung stark abhängig von der Präzision der Prompts und der Kontextintegration der eingesetzten Tools. Wie die Arbeit mit Cursor zeigte, ist insbesondere bei komplexeren Aufgaben ein dialogischer Ansatz mit Feedback-Loops und manueller Kontrolle weiterhin unverzichtbar. Dennoch zeigen sowohl Forschung als auch Praxis, dass generative KI einen spürbaren Effizienzgewinn im Entwicklungsalltag ermöglicht. Wangoo [Wan] stellt heraus, 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 verstärkte Einsatz generativer KI hat in den letzten Jahren eine Vielzahl neuer Werkzeuge und Methoden in der Softwareentwicklung etabliert. Besonders die Integration von Large Language Models (LLMs) in Entwicklungsumgebungen hat die Art, wie Entwickler*innen arbeiten, maßgeblich verändert.

Zu den wichtigsten Werkzeugen zählen unter anderem **GitHub Copilot**, **Cursor AI**, **Amazon CodeWhisperer** und **Devin AI**. Diese Tools werden in der Literatur umfassend dargestellt und spielen laut Esposito et al. [Esp] sowie Nguyen-Duc et al. [?] eine zentrale Rolle in der aktuellen Entwicklungspraxis.

GitHub Copilot wird besonders häufig eingesetzt und unterstützt Entwickler*innen bei der automatischen Codegenerierung und Vervollständigung direkt in der IDE. Esposito

et al. [Esp, S. 2] beschreiben, dass solche Werkzeuge zunehmend in frühen Phasen des Entwicklungsprozesses verwendet werden, etwa beim Übergang von Anforderungen zu Architektur oder bei der Erstellung von Code aus natürlichsprachigen Beschreibungen.

Cursor AI und ähnliche Tools ermöglichen einen dialogorientierten Workflow, bei dem nicht nur einzelne Codezeilen, sondern ganze Features, Module oder sogar Projekte automatisch erstellt und verfeinert werden können. Dabei kommen Methoden wie Prompt Engineering, Retrieval-Augmented Generation (RAG) und agentenbasierte Ansätze zum Einsatz (vgl. Esposito et al., [Esp, S. 3–4]). Karhu et al. [Kar] verweisen in ihrer Übersicht auf die Diskrepanz zwischen den hohen Erwartungen an KI-gestützte Testautomatisierung und den realen Herausforderungen bei deren Einführung in der industriellen Praxis. Aktuelle Ansätze wie von Islam und Sandborn [Isl] demonstrieren, dass multimodale generative KI zunehmend auch in der agilen Aufwandsschätzung, beispielsweise bei der automatischen Ermittlung von Story Points, eingesetzt wird.

Die Entwicklung neuer Werkzeuge und Methoden wird in der aktuellen Literatur ausführlich beschrieben. So erläutert Kerr [Ker] anhand von Praxisbeispielen die Integration von Copilot in moderne Entwicklungsprozesse. Nguyen-Duc et al. [ND] betonen die wachsende Bedeutung von Prompt Engineering und dialogbasierten Workflows. Weisz et al. [Wei] formulieren Designprinzipien für die erfolgreiche Nutzung generativer KI-Tools, während Geyer et al. [Gey] den Einfluss auf die agile Qualitätssicherung untersuchen. Gill [Gil] unterstreicht die Bedeutung von speziell angepassten, agilen Entwicklungsprozessen für KI-basierte Projekte. Die Erfahrungen aus der Praxis - wie sie etwa von Rogachev [Rog] beschrieben werden - bestätigen, dass sich durch die Kombination dieser Ansätze sowohl Produktivität als auch Codequalität nachhaltig verbessern lassen.

KI-Systeme werden dabei nicht mehr nur als Automatisierungstools verstanden, sondern zunehmend als kreative Partner, die neue Formen der kollaborativen Ideenfindung und Gestaltung ermöglichen [Kha].

Im praktischen Teil dieser Arbeit (vgl. Kapitel 3) zeigte sich, dass die Kombination dieser Werkzeuge erhebliche Produktivitätsgewinne ermöglicht, vor allem beim schnellen Prototyping, bei Standardaufgaben (Boilerplate) und bei der automatischen Generierung von Tests. Cursor AI konnte darüber hinaus durch die Möglichkeit, Kontext wie Screenshots oder Fehlermeldungen einzubinden, bei der Fehlersuche und dem Debugging zusätzliche Mehrwerte bieten.

Neben den Werkzeugen haben sich auch neue Methoden etabliert:

- **Prompt Engineering:** Entwickler*innen formulieren Anforderungen in natürlicher Sprache, die direkt von der KI interpretiert werden (vgl. Esposito et al., [Esp, S. 2–3]).

- **Retrieval-Augmented Generation (RAG):** KI-Tools kombinieren projektspezifische Kontextdaten (z. B. Dokumentation, vorhandener Code) mit aktuellen Benutzeranfragen, um passgenaue Lösungen zu generieren (vgl. Esposito et al., [Esp, S. 4]).
- **Human-in-the-Loop und Pair Programming:** Laut Nguyen-Duc et al. [ND, S. 8] und Fraunhofer IESE [Sie] wird die Zusammenarbeit von Mensch und KI (z. B. durch Feedback-Loops) immer wichtiger, um Qualität und Anpassungsfähigkeit der Entwicklung zu sichern. 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.

Im Blog von Fraunhofer IESE [Sie] wird betont, dass diese neuen Tools nicht nur als Auto vervollständigung dienen, sondern immer mehr Aufgaben im gesamten Entwicklungsprozess übernehmen – bis hin zur automatischen Erstellung von Tests und 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. Die folgenden Abschnitte beleuchten diese Herausforderungen anhand aktueller Literatur und reflektieren sie unter Einbezug der im praktischen Teil gewonnenen Erfahrungen.

5.1 Sicherheits- und Datenschutzaspekte

Die Integration generativer KI-Tools in die Softwareentwicklung eröffnet nicht nur neue Chancen, sondern schafft zugleich neue Risiken im Bereich der IT-Sicherheit und des Datenschutzes. Shi et al. [Shi] zeigen, dass KI-gestützte Entwicklung sowohl zur Erkennung und Vermeidung von Sicherheitslücken beitragen kann, aber auch neue Angriffsflächen entstehen, insbesondere wenn unsichere oder fehlerhafte Codevorschläge ungeprüft übernommen werden.

Alwageed und Khan [Alw] betonen, dass generative KI-Modelle wie LLMs dazu beitragen können, Best Practices im Bereich Security-by-Design umzusetzen. Gleichzeitig weisen sie darauf hin, dass „over-reliance on AI-generated code may lead to subtle vulnerabilities and propagate insecure coding patterns if not carefully validated by human experts“ [Alw, S. 9].

Auch Fragen des Datenschutzes stehen im Fokus: Bei der Nutzung externer KI-Modelle (z.B. in Cloud-Umgebungen) können sensible Daten unbeabsichtigt an Dritte weitergegeben werden. Shi et al. [Shi] fordern deshalb, dass Security-Reviews, Audit-Trails und eine konsequente Einbindung menschlicher Expertise („Human-in-the-Loop“) zentrale Bestandteile des Entwicklungsprozesses bleiben müssen.

Weisz et al. [Wei] betonen, dass regelmäßige Security-Reviews und der Einsatz von Human-in-the-Loop-Mechanismen zentrale Bausteine für die sichere Nutzung generativer KI sind. Studien zeigen außerdem, dass viele Angriffe darauf abzielen, gezielt das Verhalten generativer Modelle zu manipulieren. Aus diesem Grund fordern Shi et al. [Shi] und Alwageed et al. [Alw] verstärkte Kontrollmechanismen und gezielte Trainings, um die Robustheit der Systeme zu erhöhen.

Im Praxisteil dieser Arbeit zeigte sich, dass KI-Tools wie Copilot und Cursor zwar potenziell sicherheitskritische Muster (z.B. hardcodierte Passwörter) erkennen und warnen können, ihre Vorschläge jedoch stets von Entwickler*innen geprüft und angepasst werden sollten.

5.1.1 Sicherheitsrisiken durch generative Modelle

Sicherheitsrisiken durch generative Modelle

Generative KI-Modelle bringen neben allgemeinen IT-Sicherheitsfragen spezifische neue Bedrohungen mit sich. Shi et al. [Shi] beschreiben, dass Angriffe wie „Prompt Injection“, bei denen gezielt manipulierte Eingaben dazu führen, dass die KI unsicheren oder schädlichen Code generiert, eine reale Gefahr darstellen. Ebenso können sogenannte „adversarial attacks“ dazu führen, dass durch minimale Veränderungen an den Eingabedaten unerwartete und potenziell sicherheitskritische Verhaltensweisen im generativen Modell ausgelöst werden.

Aktuelle Forschung zeigt, dass der Einsatz generativer KI neue Angriffsflächen und Risiken für die IT-Sicherheit schafft. Shi et al. [Shi] weisen darauf hin, dass insbesondere sogenannte Prompt Injections und adversarielle Angriffe gezielt genutzt werden können, um unsicheren oder schädlichen Code zu erzeugen. Alwageed und Khan [Alw] betonen zudem, dass eine Übernahme von KI-generierten Vorschlägen ohne sorgfältige Überprüfung zu subtilen Sicherheitslücken führen kann. Das Fraunhofer IESE [Sie] hebt hervor, dass externe KI-Modelle entlang der Software-Supply-Chain zusätzliche Datenschutzprobleme verursachen können.

Ein weiteres Risiko besteht im sogenannten „Model Poisoning“: Hierbei werden während des Trainings gezielt fehlerhafte oder bösartige Daten eingespeist, um die KI zu manipulieren und Schwachstellen zu platzieren. Alwageed und Khan [Alw] weisen darauf hin, dass große generative Modelle wie LLMs besonders anfällig für solche Angriffe sind, da sie auf umfangreiche und oftmals öffentlich verfügbare Datenquellen zurückgreifen. In diesem Zusammenhang betonen sie die Bedeutung von regelmäßigen Security-Reviews und der Entwicklung von Abwehrmechanismen gegen adversarial attacks.

Der Blog von Fraunhofer IESE [Sie] hebt hervor, dass gerade bei der Nutzung externer KI-Modelle entlang der Software-Supply-Chain neue Risiken entstehen können. Unzureichend geprüfter oder von Dritten generierter Code kann unbemerkt Schwachstellen in den Entwicklungsprozess einschleusen.

5.2 Ethische und soziale Implikationen

Die zunehmende Integration generativer KI in den Entwicklungsalltag wirft weitreichende ethische und soziale Fragen auf. Weisz et al. [Wei] betonen, dass mit der Automatisierung von Entwicklungsaufgaben die Notwendigkeit steigt, Verantwortung und Entscheidungsprozesse klar zu definieren. Insbesondere die Nachvollziehbarkeit und Transparenz von KI-generierten Lösungen gelten als zentrale Herausforderungen im Hinblick auf ethische Standards und die Überprüfbarkeit von Ergebnissen.

Ein zentrales ethisches Problem besteht im sogenannten Bias: Generative KI-Modelle können bestehende Vorurteile aus Trainingsdaten übernehmen und reproduzieren. Wie Weisz et al. [Wei] herausstellen, müssen Entwickler:innen und Unternehmen darauf achten, geeignete Kontrollmechanismen („Guardrails“) zu etablieren, um Diskriminierung, Fehlinformationen und unfaire Vorschläge zu vermeiden.

Neben den technischen Herausforderungen gewinnen ethische und soziale Fragestellungen immer mehr an Bedeutung. Weisz et al. [Wei] unterstreichen die Notwendigkeit, bei der Entwicklung und Nutzung von KI-Systemen Transparenz, Nachvollziehbarkeit und Fairness sicherzustellen. Flores-Saviaga et al. [FS] machen zudem deutlich, dass der Aspekt der Barrierefreiheit oft noch zu wenig Beachtung findet, obwohl KI-Systeme prinzipiell neue Teilhabemöglichkeiten eröffnen könnten.

Schmitt et al. [Sch] untersuchen darüber hinaus die Auswirkungen generativer KI auf die berufliche Identität von Entwickler:innen. Ihre Studie zeigt, dass der Einsatz von KI-Tools zu Verunsicherung hinsichtlich der eigenen Rolle und Wertschätzung führen kann. Gleichzeitig ergeben sich neue Möglichkeiten zur Kompetenzentwicklung und Zusammenarbeit, sofern der Fokus auf Mensch-KI-Kollaboration liegt.

Auch im organisatorischen Kontext ergeben sich Herausforderungen: Laut Nguyen-Duc et al. [ND] sind Unternehmen gefordert, klare Leitlinien für die Nutzung von GenAI-Tools zu formulieren, um Verantwortlichkeiten, Qualitätsstandards und ethische Prinzipien verbindlich zu verankern.

5.2.1 Ethische Konflikte und Bias in KI-Systemen

Eines der zentralen ethischen Probleme beim Einsatz generativer KI in der Softwareentwicklung ist die Gefahr von Bias und Diskriminierung. Wie Weisz et al. [Wei] herausstellen, können große Sprachmodelle und generative Systeme bestehende Vorurteile, Diskriminierungen oder Stereotypen aus den Trainingsdaten übernehmen und diese im erzeugten Code oder in den Vorschlägen reproduzieren. Dies kann zu unfairen, potenziell diskriminierenden Ergebnissen führen und damit ethische Grundsätze sowie Gleichbehandlungsprinzipien verletzen.

Um solchen Risiken zu begegnen, empfehlen Weisz et al. [Wei], dass Entwickler:innen und Organisationen technische und organisatorische Maßnahmen („Guardrails“) implementieren, die sicherstellen, dass generative KI-Lösungen regelmäßig auf Fairness, Transparenz und mögliche Verzerrungen geprüft werden. Dazu zählen etwa kontrollierte Testdatensätze, Diversity-Checks oder der Einsatz spezialisierter Überwachungsmechanismen.

Schmitt et al. [Sch] weisen darauf hin, dass die Gefahr von Bias nicht nur technischer Natur ist, sondern auch soziale und organisationale Auswirkungen haben kann. Insbesondere im Kontext beruflicher Identität und Teamdynamik kann eine unkritische Nutzung von KI-Systemen zu Unsicherheiten, Vertrauensverlust und Spannungen führen – etwa wenn Vorschläge der KI als neutral oder objektiv wahrgenommen werden, obwohl sie verzerrt oder unvollständig sind.

Insgesamt machen beide Quellen deutlich, dass ethische Konflikte und der Umgang mit Bias zentrale Herausforderungen für den erfolgreichen und verantwortungsvollen Einsatz generativer KI in der Softwareentwicklung darstellen.

5.2.2 Langfristige Auswirkungen auf Entwickler:innen-Rollen

Der verstärkte Einsatz generativer KI verändert nicht nur technische Prozesse, sondern wirkt sich auch langfristig auf die Rolle von Entwickler:innen aus. Schmitt et al. [Sch] zeigen, dass der zunehmende Einsatz von KI-Tools wie Copilot oder ChatGPT zu einem Wandel der beruflichen Identität und des Selbstverständnisses von Softwareentwickler:innen führen kann. Während einerseits Routinetätigkeiten und repetitive Aufgaben zunehmend automatisiert werden, rücken Kompetenzen wie Prompt-Engineering, Systembewertung und die kritische Reflexion von KI-Ergebnissen stärker in den Vordergrund.

Die Studie von Schmitt et al. [Sch] verdeutlicht zudem, dass viele Entwickler:innen durch die Integration von GenAI neue Möglichkeiten zur Kompetenzentwicklung sehen – etwa im Bereich Mensch-KI-Kollaboration oder im Aufbau von Schnittstellenkompetenzen

zwischen Entwicklung, Domänenwissen und KI-Nutzung. Gleichzeitig berichten die Autor:innen aber auch von Verunsicherung und Identitätskonflikten, die durch die Neuverteilung von Aufgaben und die wachsende Abhängigkeit von KI-Systemen entstehen können.

Auch auf organisatorischer Ebene ergeben sich laut Nguyen-Duc et al. [ND] langfristige Veränderungen: Die Einführung von GenAI-Tools erfordert nicht nur technisches, sondern auch soziales Change Management, etwa durch die Entwicklung neuer Rollenprofile, angepasster Schulungskonzepte und überarbeiteter Verantwortlichkeiten im Team. Entscheidend ist laut beiden Quellen, dass Unternehmen und Entwickler:innen die Veränderungen aktiv gestalten und sich auf eine kontinuierliche Weiterentwicklung der beruflichen Rollen einlassen.

Im praktischen Teil dieser Arbeit zeigte sich dieser Wandel exemplarisch: Während der Implementierung des Map-Screens in der Locals-App verlagerte sich der Fokus zunehmend von der reinen Code-Implementierung hin zur Fähigkeit, die richtigen Prompts zu formulieren, generierte Vorschläge kritisch zu prüfen und die Zusammenarbeit mit KI-Tools aktiv zu gestalten. Anstelle von klassischen Routinetätigkeiten dominierten Tätigkeiten wie das Feintuning von Prompts, die Auswahl zwischen unterschiedlichen KI-generierten Lösungswegen und die Integration von Feedback in den Entwicklungsprozess.

Diese Entwicklung bestätigt die Beobachtung von Schmitt et al. [Sch], dass Entwickler:innen künftig vermehrt als Schnittstelle zwischen Mensch und KI agieren und Kompetenzen wie Systembewertung, Prompt-Engineering und kritische Reflexion an Bedeutung gewinnen.

5.2.3 Technische und organisatorische Hürden bei der Einführung von KI

Die Einführung generativer KI ist mit zahlreichen technischen und organisatorischen Hürden verbunden. Nguyen-Duc et al. [ND] betonen den Mangel an Standards, geeigneten Benchmarks und validen Testdaten als zentrale Herausforderungen. Gill [?] verweist auf die Notwendigkeit, agile Prozesse speziell für KI-Projekte weiterzuentwickeln, während das Fraunhofer IESE [?] auf Unsicherheiten im Team und fehlende Akzeptanz hinweist. Sergeyuk et al. [?] unterstreichen in ihrer Übersicht, dass auch Usability und UX-Design bei der Integration von KI-Tools stärker berücksichtigt werden müssen. Sifi [Sif] hebt hervor, dass die subjektive Nutzererfahrung bei der Einführung neuer KI-Tools oft unterschätzt wird und die Akzeptanz entscheidend von transparenter Kommunikation und kontinuierlichem Feedback abhängt.

Weitere technische Herausforderungen bestehen in der Qualitätssicherung der generierten Ergebnisse. Laut Nguyen-Duc et al. [ND] ist es 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.

Mit der fortschreitenden Integration von KI in die Softwareentwicklung entstehen nicht nur neue technische Herausforderungen, sondern auch veränderte Anforderungen an Sicherheit, Arbeitsorganisation und langfristige Kollaborationsmodelle [Haz]

Auf organisatorischer Ebene betonen sowohl Nguyen-Duc et al. [ND] als auch Schmitt et al. [Sch], dass fehlende Akzeptanz und Unsicherheit im Team, unklare Verantwortlichkeiten sowie mangelnde Schulung zu erheblichen Implementierungsbarrieren führen können. Die Umstellung auf KI-gestützte Prozesse erfordert häufig ein umfassendes Change Management, die Anpassung bestehender Arbeitsweisen und neue Formen der Zusammenarbeit. Schmitt et al. [Sch] weisen darauf hin, dass die erfolgreiche Einführung von GenAI-Tools nicht nur technisches Know-how, sondern auch kulturelle Offenheit und kontinuierliche Weiterbildung im Team voraussetzt. Richards und Wessel [Ric] argumentieren, dass für den Erfolg conversationaler KI-Assistenzsysteme eine enge Zusammenarbeit von HCI- und KI-Forschung erforderlich ist, um praxisnahe Evaluationsmethoden zu etablieren.

6 Wirtschaftliche und gesellschaftliche Auswirkungen

Die Integration generativer KI in die Softwareentwicklung wirkt sich nicht nur auf technischer Ebene, sondern zunehmend auch auf wirtschaftliche Strukturen und gesellschaftliche Prozesse aus. Die folgenden Abschnitte analysieren, wie sich Unternehmen, Arbeitsmärkte und die Rolle von Softwareentwickler:innen im Zuge der KI-Einführung verändern. Im Zentrum stehen Fragen nach Produktivität, Wertschöpfung, Arbeitsplatzstruktur und den Chancen und Risiken für Unternehmen und Gesellschaft.

6.1 Veränderungen in Softwareunternehmen

Mit der Verbreitung generativer KI-Technologien stehen Softwareunternehmen vor einem tiefgreifenden Wandel. Marguerit [Mara] beschreibt, dass Unternehmen zunehmend KI-basierte Automatisierungslösungen einsetzen, um Entwicklungsprozesse zu beschleunigen und Ressourcen effizienter zu nutzen. Diese Entwicklung führt dazu, dass klassische Rollenmodelle und Teamstrukturen neu bewertet werden müssen.

Farach et al. [Far] argumentieren, dass digitale Arbeit durch KI-Tools einen eigenständigen Produktionsfaktor darstellt, der traditionelle Vorstellungen von Arbeitsteilung und Wertschöpfung grundlegend verändert. In vielen Unternehmen werden Aufgaben wie das Schreiben von Standardcode, Testing oder das Generieren von Dokumentation zunehmend automatisiert, während der Fokus auf kreative, überwachende und strategische Tätigkeiten wächst.

Rothschild et al. [Rot] diskutieren das Konzept der „agentischen Ökonomie“, in der generative KI-Systeme zunehmend eigenständig Entscheidungen treffen und so ganze Wertschöpfungsketten transformieren. Unternehmen sind laut den Autoren gefordert, ihre Prozesse kontinuierlich an diese neue Form digitaler Arbeit und Delegation anzupassen.

Storey et al. [Sto] betonen, dass die Einführung generativer KI-Tools nicht nur ökonomische, sondern auch organisatorische Anpassungen erfordert. Unternehmen müssen neue Kompetenzen fördern, Veränderungsbereitschaft unterstützen und klare ethische

sowie regulatorische Leitplanken setzen, um Risiken zu minimieren und die Akzeptanz im Team zu erhöhen.

Die Literatur macht deutlich, dass der Erfolg von KI-Integration maßgeblich davon abhängt, inwieweit Unternehmen nicht nur in Technologie, sondern auch in Weiterbildung, Organisationsentwicklung und einer offenen Innovationskultur investieren.

6.2 Auswirkungen auf den Arbeitsmarkt und Entwickler:innen-Rollen

Die Verbreitung generativer KI wirkt sich bereits heute spürbar auf den Arbeitsmarkt für Softwareentwickler:innen aus. Marguerit [Mara] argumentiert, dass KI nicht nur Automatisierung, sondern auch eine qualitative Veränderung von Arbeit mit sich bringt: Während repetitive Tätigkeiten und Routinetasks zunehmend automatisiert werden, entstehen neue Aufgabenfelder rund um die Steuerung, Überwachung und das Feintuning von KI-basierten Systemen.

Ahmadi et al. [Ahm] zeigen anhand von Analysen aktueller Jobanzeigen, dass Kompetenzen im Umgang mit Tools wie ChatGPT, Copilot und anderen generativen KI-Anwendungen zunehmend nachgefragt werden. Dies deutet darauf hin, dass die Nachfrage nach klassischen Programmierkenntnissen zwar bestehen bleibt, aber zunehmend durch Fähigkeiten im Prompt-Engineering, KI-Management und in der Bewertung KI-generierter Ergebnisse ergänzt wird.

Farach et al. [Far] sehen in der digitalen Arbeit mit KI-Tools einen neuen Produktionsfaktor, der es Unternehmen ermöglicht, Entwicklungsleistungen flexibler und globaler zu organisieren. Zugleich betonen sie, dass die Einführung generativer KI auch zu Unsicherheiten auf dem Arbeitsmarkt führen kann, etwa durch die Verlagerung von Aufgaben, Veränderungen im Qualifikationsprofil oder mögliche Substitutionseffekte bei stark standardisierten Tätigkeiten.

Storey et al. [Sto] machen darauf aufmerksam, dass der Wandel nicht nur technische, sondern auch soziale Kompetenzen erfordert. Die Fähigkeit, mit KI-Systemen kollaborativ zu arbeiten, ethische Risiken zu erkennen und verantwortungsvoll mit automatisierten Vorschlägen umzugehen, wird zu einer Schlüsselkompetenz in modernen Entwicklerteams. Aktuelle Studien zeigen, dass generative KI-Tools die Arbeitsmuster von Wissensarbeiter:innen spürbar verändern und insbesondere Aufgaben mit hohem Automatisierungspotenzial nachhaltig beeinflussen [Dil].

Marguerit [Mara] weist zudem darauf hin, dass sich nicht nur die Art der Arbeit, sondern auch Beschäftigungsstrukturen und Lohngefüge im Zuge der Automatisierung durch KI

deutlich wandeln werden. Die langfristigen Auswirkungen auf die Arbeitsplatzsicherheit und berufliche Entwicklung sind dabei weiterhin Gegenstand intensiver Forschung.

6.3 Zukunftsperspektiven und strategische Empfehlungen

Die wissenschaftliche Literatur macht deutlich, dass der Einsatz generativer KI in der Softwareentwicklung noch am Anfang einer langfristigen Transformationsphase steht. Storey et al. [Sto] sehen großes Potenzial für Unternehmen, sich durch frühzeitige Investitionen in KI-Kompetenzen, datenbasierte Prozesse und ethische Leitlinien Wettbewerbsvorteile zu sichern. Marguerit [Mara] betont, dass kontinuierliche Weiterbildung und die Entwicklung neuer Rollenprofile entscheidend sind, damit Beschäftigte mit dem technologischen Wandel Schritt halten können. Auch nach Einschätzung von IBM spielt der zunehmende Einsatz von KI in der Softwareentwicklung eine Schlüsselrolle für die Transformation von Entwicklungsprozessen und die Steigerung von Effizienz sowie Innovationskraft [A].

Habibi [Hab] unterstreicht die Bedeutung von Open-Source-Ansätzen und kollaborativen Entwicklungsmodellen, um Innovation und Transparenz im KI-Ökosystem zu fördern. McNamara und Marpu [McN] weisen darauf hin, dass Unternehmen verstärkt auf flexible, adaptive Strukturen setzen müssen, um auf die exponentiell steigende Geschwindigkeit technologischer Veränderungen reagieren zu können.

Storey et al. [Sto] sehen in der kontinuierlichen Weiterentwicklung von Informationssystemen eine zentrale Chance, gesellschaftliche Teilhabe und Innovation zu fördern. Sie argumentieren, dass die gezielte Nutzung generativer KI nicht nur ökonomische, sondern auch soziale und kreative Potenziale freisetzen kann - vorausgesetzt, die Integration erfolgt verantwortungsvoll und unter Berücksichtigung ethischer Leitlinien. Mit der verstärkten Nutzung generativer KI in der Softwareentwicklung gewinnen Fragen zu Lizenzierung, Urheberrecht und Datenverantwortung weiter an Bedeutung [Sta].

Alanoca et al. [Alab] bieten mit ihrer Taxonomie für globale KI-Regulierung einen Rahmen, der Unternehmen und Gesellschaften Orientierung im Umgang mit generativen KI-Technologien geben kann. Gerade vor dem Hintergrund unterschiedlicher rechtlicher Anforderungen wird die Ausgestaltung entsprechender Leitplanken zu einer strategischen Aufgabe für alle Beteiligten.

Die Autoren sind sich einig, dass die Integration von GenAI-Tools nur dann nachhaltig gelingt, wenn Unternehmen strategisch in Kompetenzen, Change Management und eine offene Innovationskultur investieren. Farach et al. [Far] sehen zudem in der Betrachtung digitaler Arbeit als eigenständigen Produktionsfaktor einen wichtigen Schritt, um die

Wertschöpfungspotenziale von KI systematisch zu erschließen und zugleich soziale Risiken abzufedern.

6.4 Kosten-Nutzen-Analyse von KI-gestützter Softwareentwicklung

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

Allerdings entstehen auch neue Investitionen: Die Einführung und Wartung von KI-Systemen erfordert gezielte Weiterbildung, Anpassung von Infrastrukturen und häufig eine Neuausrichtung bestehender Entwicklungsprozesse. Habibi [Hab] betont, dass proprietäre KI-Lösungen mit Lizenzgebühren und laufenden Betriebskosten verbunden sind, während Open-Source-Modelle zwar günstiger sein können, dafür aber einen höheren Initialaufwand für Customizing und Integration erfordern. Song et al. [Son] zeigen anhand von Open-Source-Projekten, dass KI-basierte Assistenzsysteme die Kollaboration und Codequalität nicht nur steigern, sondern auch zu einer Demokratisierung des Entwicklungsprozesses beitragen können.

McNamara und Marpu [McN] verweisen darauf, dass der Nutzen von KI-gestützter Entwicklung maßgeblich von der Fähigkeit der Unternehmen abhängt, strategische Ziele, Kostenstruktur und Wertschöpfungspotenziale aufeinander abzustimmen. Storey et al. [Sto] ergänzen, dass zu den nicht-monetären Nutzenaspekten vor allem Flexibilität, Innovationspotenzial und die Gewinnung von Wettbewerbsvorteilen zählen. Umgekehrt ist zu beachten, dass Fehlinvestitionen, Qualitätsprobleme bei KI-generiertem Code und ethische Risiken zu erheblichen Folgekosten führen können, wenn diese nicht frühzeitig adressiert werden.

Anantrasirichai et al. [Ana] betonen, dass die fortschreitende Integration von KI-Technologien auch die kreativen Branchen nachhaltig verändert und neue Formen gesellschaftlicher Teilhabe ermöglicht. Die Rolle von KI als Impulsgeber für Innovation wird in Zukunft weiter an Bedeutung gewinnen.

Eine sorgfältige Kosten-Nutzen-Abwägung und eine ständige Anpassung der Strategie sind daher zentrale Voraussetzungen für den erfolgreichen und nachhaltigen Einsatz von generativer KI in der Softwareentwicklung.

6.5 Fazit

Die Analyse der wirtschaftlichen und gesellschaftlichen Auswirkungen generativer KI in der Softwareentwicklung zeigt: Während Unternehmen und Entwickler:innen deutliche Effizienzgewinne, neue Wertschöpfungsmodelle und innovative Arbeitsformen erschließen können, entstehen zugleich neue Herausforderungen für Arbeitsmärkte, Organisationssstrukturen und Qualifikationsprofile. Die nachhaltige Integration von KI-Tools erfordert daher nicht nur technologische Investitionen, sondern vor allem eine strategische Anpassung von Unternehmensstrukturen, kontinuierliche Weiterbildung und die Etablierung ethischer Leitlinien. Nur so lassen sich die Potenziale generativer KI langfristig nutzen und soziale Risiken wirksam begrenzen.

7 Fazit und Ausblick

7.1 Zusammenfassung der Kernergebnisse

Die vorliegende Arbeit zeigt, dass generative KI-Tools wie GitHub Copilot, Cursor oder Devin die Softwareentwicklung nachhaltig verändern. Zentrale Erkenntnisse sind signifikante Effizienzsteigerungen, die Automatisierung vieler Routineaufgaben und die Einführung neuer Werkzeuge und Methoden im Entwicklungsalltag. Die praktische Demonstration verdeutlichte, wie sich durch KI-basierte Assistenzsysteme sowohl Entwicklungszeit als auch Einstiegshürden verringern lassen – etwa bei der Initialisierung neuer Komponenten oder beim Refactoring von Code.

Gleichzeitig offenbarten sich aber auch klare Grenzen und Herausforderungen: Die Qualität von KI-generierten Vorschlägen ist abhängig von präzisen Prompts, domänen-spezifischem Kontext und der kritischen Überprüfung durch Entwickler:innen. Unsicherheiten entstehen insbesondere bei sicherheitskritischen Anwendungen, bei denen KI-Modelle fehleranfällige oder nicht nachvollziehbare Lösungen produzieren können. Zudem wurde deutlich, dass ethische Fragen wie der Umgang mit Bias, Fairness sowie Nachvollziehbarkeit und Verantwortung nicht allein durch die Technik gelöst 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

1. Wie verändert generative KI traditionelle Entwicklungspraktiken in der Softwareentwicklung?

Generative KI-Tools automatisieren viele bislang manuelle und repetitive Tätigkeiten wie Code-Generierung, Testing und Dokumentation. Dadurch verschieben sich die Schwerpunkte der Entwicklungsarbeit hin zu Prompt-Engineering, Qualitätskontrolle und Systembewertung. Entwicklungspraktiken werden iterativer, dialogorientierter und stärker von Mensch-KI-Kollaboration geprägt.

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

Zu den größten Herausforderungen zählen neue Angriffsflächen (z. B. durch fehlerhafte KI-generierte Vorschläge), ethische Risiken wie Bias und Diskriminierung sowie Fragen der Nachvollziehbarkeit, Verantwortung und Wartbarkeit von Code. Regelmäßige Überprüfung, Peer-Reviews und klare Leitlinien sind für den sicheren Einsatz unerlässlich.

3. Wie unterstützt generative KI Softwareentwickler in agilen Entwicklungsprozessen?

Generative KI-Tools fördern schnellere Iterationen, erleichtern die Anpassung von Anforderungen und bieten Hilfestellungen beim Testen und Debugging. Sie beschleunigen den Entwicklungszyklus, unterstützen das Prototyping und ermöglichen eine engere Verzahnung von Entwicklung, Test und Deployment – insbesondere in agilen Teams.

4. 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 zeigte, dass Tools wie Copilot, Cursor oder Bolt die Entwicklungszeit für typische Aufgaben (z. B. Erstellung von Komponenten, Einbindung von APIs) deutlich reduzieren können. Gleichzeitig hängt die Codequalität stark von der kritischen Bewertung, Anpassung und Überprüfung der KI-Vorschläge durch die Entwickler:innen ab.

7.3 Handlungsempfehlungen für Praxis und Unternehmen

Unternehmen sollten die Einführung generativer KI-Tools strategisch planen und als kontinuierlichen Veränderungsprozess verstehen. Neben Investitionen in technische Infrastruktur ist insbesondere die Aus- und Weiterbildung der Beschäftigten ein Schlüsselfaktor für den nachhaltigen Erfolg. Ein gezieltes Training in Prompt-Engineering,

das Verständnis von KI-Modellen sowie regelmäßige Schulungen zu Sicherheits- und Ethikstandards sind essenziell.

Es empfiehlt sich, den Einsatz von KI stets mit klaren Qualitätskontrollen und Feedback-Loops zu begleiten: Mensch-KI-Kollaboration sollte durch Peer-Review-Prozesse, Human-in-the-Loop-Mechanismen und eine offene Fehlerkultur gestärkt werden. Unternehmen profitieren zudem von der Einführung transparenter Richtlinien und der Definition von Verantwortlichkeiten für den Umgang mit KI-basierten Lösungen.

Zudem sollte der Dialog zwischen Entwickler:innen, Management und betroffenen Stakeholdern aktiv gefördert werden, um Akzeptanzprobleme und Widerstände frühzeitig zu adressieren. Die Schaffung einer innovationsfreudlichen Unternehmenskultur, die Experimente mit neuen Werkzeugen erlaubt und Lernprozesse unterstützt, ist eine zentrale Voraussetzung für die erfolgreiche Integration von KI in die Entwicklungsarbeit.

7.4 Einschränkungen der Arbeit

Die vorliegende Untersuchung basiert auf einer literaturgestützten Analyse und einer exemplarischen praktischen Demonstration. Sie erhebt keinen Anspruch auf empirische Generalisierbarkeit und kann daher keine abschließenden Aussagen zur Wirksamkeit aller KI-Tools in jeder Unternehmenskonstellation treffen. Die rechtlichen und gesellschaftspolitischen Dimensionen der KI-Nutzung konnten nur angeschnitten werden; ebenso wurden regionale Unterschiede und branchenspezifische Besonderheiten weitgehend ausgeklammert. Die Praxiserfahrung beschränkt sich zudem auf ausgewählte Tools und einen konkreten Anwendungskontext (Entwicklung einer Map-Komponente in React Native).

7.5 Offene Fragen und Forschungsbedarf

Viele offene Fragen bleiben für die Zukunft bestehen. Insbesondere der langfristige Einfluss generativer KI auf Arbeitsmarkt, Teamstrukturen und die Entwicklung komplexer Softwaresysteme ist noch nicht abschließend erforscht. Wichtige Forschungsthemen sind:

- Wie verändert sich die Qualität und Wartbarkeit von Code bei zunehmender KI-Unterstützung?
- Welche ethischen, rechtlichen und sicherheitstechnischen Standards sind für den flächendeckenden Einsatz von KI in der Softwareentwicklung notwendig?

- Wie lässt sich die Zusammenarbeit zwischen Mensch und KI so gestalten, dass Kreativität, Transparenz und Verantwortlichkeit erhalten bleiben?
- In welchem Maße wird KI künftig als Entscheidungsinstanz und nicht nur als Assistenzsysteem agieren?

Treude und Storey [Tre] fordern eine Neuausrichtung der empirischen Softwaretechnik hin zu einem Paradigma, das die Möglichkeiten und Limitationen von KI in den Mittelpunkt methodischer Innovationen rückt.

7.6 Ausblick

Die nächsten Jahre werden von einer weiteren Durchdringung der Softwareentwicklung durch generative KI geprägt sein. Unternehmen, die frühzeitig auf eine intelligente Verzahnung von Mensch und KI setzen, können Innovationspotenziale optimal ausschöpfen und ihre Wettbewerbsfähigkeit ausbauen. Gleichzeitig bleibt der gesellschaftliche und ethische Diskurs von zentraler Bedeutung: Der nachhaltige und verantwortungsvolle Umgang mit KI erfordert klare Leitplanken, kontinuierliche Qualifizierung und die Bereitschaft, gewachsene Rollen und Prozesse im Entwicklungsalltag immer wieder neu zu hinterfragen.

Die Softwareentwicklung steht damit exemplarisch für den umfassenden Wandel der Arbeitswelt im KI-Zeitalter.

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
- [Alaa] 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>
- [Alab] ALANOCA, Sacha; GUR-ARIEH, Shira; ZICK, Tom und KLYMAN, Kevin: Comparing Apples to Oranges: A Taxonomy for Navigating the Global Landscape of AI Regulation, URL <http://arxiv.org/abs/2505.13673>
- [Alw] ALWAGEED, Hathal S und KHAN, Rafiq Ahmad: The Role of Generative AI in Strengthening Secure Software Coding Practices: A Systematic Perspective
- [Ana] ANANTRASIRICHAJ, 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>
- [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>

- [Dil] DILLON, Eleanor Wiske; JAFFE, Sonia; IMMORLICA, Nicole und STANTON, Christopher T.: Shifting Work Patterns with Generative AI, URL <http://arxiv.org/abs/2504.11436>
- [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; ROSEMBERG, 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, Nirmendra: 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, Zheyding; 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
- [Rot] ROTHSCHILD, David M.; MOBIUS, Markus; HOFMAN, Jake M.; DILLON, Eleanor W.; GOLDSTEIN, Daniel G.; IMMORLICA, Nicole; JAFFE, Sonia; LUCIER, Brendan; SLIVKINS, Aleksandrs und VOGEL, Matthew: The Agentic Economy, URL <http://arxiv.org/abs/2505.15799>
- [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, Malieh: 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://xplore.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

- [Sta] STALNAKER, Trevor; WINTERSGILL, Nathan; CHAPARRO, Oscar; HEYMANN, Laura A.; PENTA, Massimiliano Di; GERMAN, Daniel M. und POSHYVANYK, Denys: Developer Perspectives on Licensing and Copyright Issues Arising from Generative AI for Software Development, URL <http://arxiv.org/abs/2411.10877>
- [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

3.1	Ausgangszustand der Anwendung vor dem Einsatz von Copilot. <i>Copilot-Demo</i>	21
3.2	Rückmeldung und Hinweise von Copilot während der Implementierung. <i>Copilot-Demo</i>	22
3.3	Erste lauffähige Version des MapScreens nach KI-gestützter Entwicklung. <i>Copilot-Demo</i>	23
3.4	Event-Details und Callouts auf dem MapScreen. <i>Copilot-Demo</i>	23
3.5	Ausgangszustand der Anwendung vor Einsatz von Cursor. <i>Cursor Demo</i>	24
3.6	Typische Fehlermeldung und autonomes Ausführen von Terminalbefehlen durch Cursor beim Einrichten des MapScreens. <i>Cursor-Demo</i>	25
3.7	Erste Umsetzungsschritte nach Bereitstellung des Kontexts und initialem Prompt. <i>Cursor-Demo</i>	26
3.8	MapScreen in der finalen Implementierung mit Cursor – zwei verschiedene Zustände/Ansichten. <i>Cursor-Demo</i>	27
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>	28
3.10	Bolt reagiert interaktiv auf Prompts, führt Terminalbefehle aus und gibt strukturiertes Feedback im Interface. <i>bolt-Demo</i>	29
3.11	Finale Web-Umsetzung: Interaktiver MapScreen mit Event-Details, Filteroptionen und Callouts. <i>bolt-Demo</i>	30

Tabellenverzeichnis

Listings

A Anhang 1

B Anhang 2