THM | CAMPUS GIESSEN | MNI
Mathematik, Naturwissenschaften und Informatik
TECHNISCHE HOCHSCHULE MITTELHESSEN

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

29. Juni 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 Science[2] 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 29. Juni 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.sciencedirect.com/journal/theoretical-computer-science/publish/guide-for-authors`

Inhaltsverzeichnis

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 [**?**].

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. [**?**]) . 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 Aufbau der Arbeit

Die Arbeit gliedert sich in folgende Kapitel:

- **Kapitel 1: Einleitung**
  - Darstellung von Hintergrund, Motivation, Zielsetzung, Forschungsfragen, methodischer Vorgehensweise und Abgrenzung.
- **Kapitel 2: Theoretische Grundlagen**
  - Definition und Funktionsweise von generativer KI in der Softwareentwicklung
  - Übersicht relevanter KI-Modelle, Algorithmen und Beispiele für KI-gestützte Entwicklungswerkzeuge
- **Kapitel 3: Praktische Demonstration**
  - Vorstellung des Projekts "Localsünd dessen Architektur
  - Implementierung einer interaktiven Kartenansicht in der React Native-Anwendung mithilfe generativer KI-Technologien
  - Darstellung der Implementierungsschritte, Code-Beispiele und erste Evaluationsergebnisse
- **Kapitel 4: Chancen durch KI**
  - Effizienzsteigerung und Automatisierung
  - Neue Werkzeuge und Methoden
  - Verbesserte Code-Qualität und Fehlerminimierung
  - Einfluss von KI auf agile Entwicklungsmethoden
  - Bezugnahme auf die Erkenntnisse aus Kapitel 3
- **Kapitel 5: Herausforderungen durch KI**

  - Sicherheits- und Datenschutzaspekte
  - Ethische Implikationen und Bias in KI-Modellen
  - Langfristige Auswirkungen auf Entwickler:innen-Rollen
  - Organisatorische und technologische Hürden
  - Risiken durch Abhängigkeit von KI-generiertem Code
  - Analyse der in Kapitel 3 möglicherweise aufgetretenen Herausforderungen und Problematiken
- **Kapitel 6: Wirtschaftliche und gesellschaftliche Auswirkungen**
  - Veränderungen in Softwareunternehmen
  - Auswirkungen auf den Arbeitsmarkt und Entwickler:innen-Rollen
  - Zukunftsperspektiven und strategische Empfehlungen
  - Kosten-Nutzen-Analyse von KI-gestützter Softwareentwicklung
- **Kapitel 7: Fazit und Ausblick**
  - Zusammenfassung der theoretischen und praktischen Erkenntnisse
  - Diskussion offener Forschungsfragen
  - Ableitung von Handlungsempfehlungen und Ausblick auf zukünftige Entwicklungen

1.5 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) hat sich in den letzten Jahren zu einer Schlüsseltechnologie in der Softwareentwicklung entwickelt. Eine anerkannte und aktuelle Definition liefert die Europäische Union. Demnach bezeichnet

> „Künstliche Intelligenz (KI) [...] ein maschinengestütztes System, das mit unterschiedlichem Grad an Autonomie für ausdrücklich oder implizit gesetzte Ziele Vorhersagen, Empfehlungen oder Entscheidungen generiert, die physische oder virtuelle Umgebungen beeinflussen können."

[noa]

Diese Definition orientiert sich an der aktuellen europäischen Gesetzgebung und wird in der wissenschaftlichen Diskussion zunehmend als Standardbegriff verwendet.

Die Entwicklung von KI-Systemen unterscheidet sich deutlich von klassischen softwarebasierten Ansätzen. Während erste KI-gestützte Werkzeuge lediglich grundlegende Aufgaben wie Syntaxprüfung oder Code-Formatierung unterstützten, übernehmen moderne Generative-AI-Tools Aufgaben, die maßgeblichen Einfluss auf das Design, die Entwicklung und Wartung von Software haben. Dazu zählen die Generierung von Code-Snippets, ganzen Funktionen oder Modulen, die Unterstützung bei der Erstellung von Unit-Tests sowie die Automatisierung von Deployment-Prozessen [Don].

In modernen generativen KI-Systemen kommen unterschiedliche Modellarchitekturen zum Einsatz, die als Grundlage für viele aktuelle Anwendungen dienen. Donvir et al. [Don] nennen in ihrer Übersicht insbesondere Transformer-Modelle (wie Large Language Models, z.B. GPT), Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs) und Diffusion Models als zentrale Modelltypen:

> „They also present a taxonomy of generative AI models, such as Generative Adversarial Networks (GANs), variational autoencoders (VAEs), and transformers, which are foundational to many modern GenAI applications."

[Don, S. 2]

Transformer-Modelle, insbesondere Large Language Models wie GPT, bilden die Grundlage für viele KI-gestützte Text- und Codegenerierungswerkzeuge (z.B. ChatGPT, GitHub Copilot oder Bard). GANs werden häufig für die Generierung von Bildern, Videos oder anderen Medieninhalten verwendet und stellen einen wichtigen Baustein generativer Anwendungen dar. Variational Autoencoders (VAEs) sind vor allem in der Datengenerierung und bei der effizienten Verarbeitung komplexer Eingabedaten relevant. Diffusion Models wiederum kommen insbesondere in der Bildgenerierung zum Einsatz und sind Grundlage moderner Tools wie Stable Diffusion. Donvir et al. [Don] heben hervor, dass diese Modelle die Vielfalt und Leistungsfähigkeit generativer KI-Systeme wesentlich bestimmen und die Entwicklung neuer Softwarewerkzeuge und -anwendungen maßgeblich vorantreiben:

> „...the paper Advancements in Generative AI: A Comprehensive Review of GANs, GPT, Autoencoders, Diffusion Model, and Transformers [5] explores state-of-the-art GenAI models that power tools like ChatGPT, Bard, and Stable Diffusion. This review highlights the wide range of capabilities these models enable, from text generation to image creation, and emphasizes their applications in software development."

[Don, S. 2]

Die kontinuierliche Weiterentwicklung dieser Modellarchitekturen bedingt auch die rasante Entwicklung neuer Werkzeuge und Methoden, die aktuelle Trends im Bereich der KI-gestützten Softwareentwicklung bestimmen. Insbesondere Generative KI (GenAI) prägt laut Esposito et al. [Esp] „die Art und Weise, wie Entwickler Code entwerfen, schreiben und warten, grundlegend". Besonders hervorzuheben ist der breite Einsatz von Large Language Models (LLMs) wie den OpenAI GPT-Modellen, die vorrangig in frühen Phasen des Softwareentwicklungszyklus, etwa im Übergang von Requirements zu Architektur und von Architektur zu Code, Anwendung finden.

Entscheidungsunterstützung und die Rekonstruktion von Softwarearchitekturen gehören zu den dominanten Anwendungsfeldern, wobei Methoden wie Few-Shot-Prompting und Retrieval-Augmented Generation (RAG) etabliert sind. Ein zentrales Merkmal aktueller Ansätze ist der weiterhin hohe Grad an menschlicher Interaktion und Validierung, wodurch vollständig autonome KI-gestützte Architekturentscheidungen bislang selten sind [Esp, S. 2, 10]. Zu den Herausforderungen gehören insbesondere die Sicherstellung von Präzision und Verlässlichkeit der Modelle, der Umgang mit Halluzinationen und ethischen Fragestellungen sowie der Mangel an domänenspezifischen Benchmarks und Evaluationsstandards [Esp, S. 2, 16].

Quellen:

- Verordnung (EU) 2024/1689 des Europäischen Parlaments und des Rates [noa]
- Donvir, A. et al. (2024): *The Role of Generative AI Tools in Application Development: A Comprehensive Review of Current Technologies and Practices* [Don]
- Esposito, M. et al. (2025): *Generative AI for Software Architecture. Applications, Trends, Challenges, and Future Directions* [Esp]

2.1.1 Beispiele für generative KI-Tools in der Praxis

Generative KI-Tools sind heute in der Softwareentwicklung weit verbreitet und decken ein breites Spektrum an Aufgaben ab – von der Code-Vervollständigung über automatische Testgenerierung bis hin zur Unterstützung ganzer Entwicklungsprojekte. Im Folgenden werden vier prominente Tools vorgestellt, die in der Praxis besonders relevant sind: **GitHub Copilot**, **TabNine**, **Cursor AI** und **Devin AI**.

GitHub Copilot GitHub Copilot ist ein KI-basierter Codeassistent, der Entwicklern direkt im IDE-Kontext Vorschläge für Code-Snippets, ganze Funktionen und sogar Tests macht. Die Integration erfolgt beispielsweise in Visual Studio Code, IntelliJ oder Eclipse. Copilot basiert auf dem OpenAI Codex-Modell und kann mit Kommentaren oder natürlicher Sprache gesteuert werden. Typische Einsatzfelder sind das schnelle Prototyping, die Generierung von Boilerplate-Code, aber auch die Hilfestellung für Einsteiger und Onboarding-Prozesse.

> „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]

Praxiserfahrung zeigt, dass GitHub Copilot die Entwicklung etwa einer React-Anwendung deutlich beschleunigen kann, indem es Codevorschläge für Authentifizierung, Routing und Formularvalidierung generiert und Fehlerbehebung unterstützt. Entwickler betonen, dass der Review und die Überprüfung der generierten Vorschläge dennoch unerlässlich bleiben [Ker].

TabNine TabNine ist ein weiteres KI-gestütztes Tool zur Code-Vervollständigung, das ursprünglich auf GPT-2 basierte und heute ein eigenes Modell nutzt. Es kann Codevorschläge in Echtzeit für verschiedene Sprachen machen und passt sich über die Zeit an den Coding-Stil des jeweiligen Entwicklers an. TabNine unterstützt alle gängigen IDEs und bietet eine Chat-Funktion, um gezielt Code-Fragen zu stellen. Besonders geschätzt wird die Flexibilität durch lokale und cloudbasierte Modelle sowie die Anpassung an unterschiedliche Datenschutzanforderungen [Don, S. 9].

Cursor AI Cursor AI repräsentiert die nächste Generation von KI-Entwicklungstools. Es ist in der Lage, auf Basis natürlicher Sprache ganze Applikationen zu generieren, nutzt Techniken wie Retrieval-Augmented Generation (RAG) und Agentic AI und kann selbstständig Code verfeinern und Projekte strukturieren. Der Fokus liegt auf End-to-End-Entwicklung und vollständiger Projektgenerierung, was insbesondere für Prototyping oder den schnellen Aufbau komplexer Softwarelösungen geeignet ist.

> „Cursor AI can generate the entire codebase of the application from the feature description of the project provided in natural language. It uses advanced AI concepts such as Retrieval Augmented Generation (RAG), Agentic AI, and prompt chaining to achieve its objectives and provides a high degree of automation in software development.“ [Don, S. 10]

Devin AI Devin AI geht noch einen Schritt weiter und bezeichnet sich selbst als „AI Software Engineer“. Devin kann komplette Softwareprojekte auf Basis von Anforderungen in natürlicher Sprache umsetzen, das Projekt in einzelne Aufgaben herunterbrechen, automatisiert testen und sogar Deployment-Skripte erstellen. Ein wesentliches Merkmal ist die Fähigkeit zur langfristigen Planung und zur kontinuierlichen Anpassung an neue Anforderungen [Don, S. 11].

Typische Einsatzszenarien

Die beschriebenen Tools werden in der Praxis in unterschiedlichen Bereichen eingesetzt:

- **Code-Generierung und Vervollständigung:** Automatisiertes Schreiben von Code, Vorschläge für Funktionen, Klassen, API-Integration etc.
- **Test- und Debugging-Unterstützung:** Generierung von Unit- und Integrationstests, Identifikation 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, S. 11].

Vorteile und Grenzen in der Praxis

Die Integration generativer KI-Tools führt zu Zeitersparnis, konsistenterem Code und einer schnelleren Einarbeitung neuer Entwickler. Gleichzeitig sind Review-Prozesse und ein kritischer Umgang mit den generierten Vorschlägen unerlässlich, um Qualitäts- und Sicherheitsrisiken zu minimieren. Fortgeschrittene Tools wie Cursor AI oder Devin AI bieten ein hohes Maß an Automatisierung, sind aber oft kostenintensiv und noch nicht für alle Anwendungsfälle ausgereift [Don, S. 13].

Quellen:

- Donvir, A. et al. (2024): *The Role of Generative AI Tools in Application Development: A Comprehensive Review of Current Technologies and Practices* [Don]
- Kerr, K. (2025): *GitHub for Beginners: Building a React App with GitHub Copilot - The GitHub Blog* [Ker]

2.1.2 Wichtige Algorithmen und Modelle in der Softwareentwicklung

Die Integration generativer Künstlicher Intelligenz (KI) in der Softwareentwicklung basiert maßgeblich auf fortschrittlichen Modellen und Algorithmen. Im Zentrum stehen insbesondere Large Language Models (LLMs) und Transformer-Architekturen, die als Grundlage moderner Coding-Tools wie GitHub Copilot, Cursor oder v0 dienen. Im Folgenden werden die wichtigsten Modelle, deren Funktionsweise und deren Bedeutung für typische Softwareentwicklungsaufgaben erläutert.

Grundprinzipien und Funktionsweise moderner KI-Modelle Large Language Models (LLMs), wie beispielsweise GPT-4 oder Code Llama, sind tiefenlernende neuronale Netze, die auf sehr großen Mengen von Quellcode und natürlicher Sprache trainiert wurden. Sie nutzen Transformer-Architekturen, die durch sogenannte Self-Attention-Mechanismen Kontextinformationen über lange Sequenzen hinweg erfassen können. Dies ermöglicht es ihnen, sowohl syntaktisch als auch semantisch komplexe Strukturen – wie etwa Code-Logik oder Designmuster – zu erkennen und zu generieren [ND, Esp].

Diffusionsmodelle spielen hingegen vor allem in der Bild- und Grafikgenerierung eine Rolle und sind für den textbasierten Softwareentwicklungsprozess bislang von unterge-

ordneter Bedeutung. Für Aufgaben wie Codegenerierung und Architektur-Design sind LLMs und Transformer-Modelle maßgeblich [Wei].

Beispiele für KI-Modelle in Coding-Tools Moderne Coding-Assistenzsysteme wie *GitHub Copilot*, *Cursor* und *v0* basieren auf Varianten großer Sprachmodelle, meist speziell auf Programmcode vortrainiert (z. B. OpenAI Codex, Code Llama, StarCoder). Diese Modelle können anhand des jeweiligen Kontexts (z. B. bestehender Code, Kommentare, Projekthistorie) automatisiert neue Code-Abschnitte generieren, Vorschläge zur Fehlerbehebung machen oder Testfälle entwerfen [Cou, Esp].

Typische Anwendungsbereiche sind:

- **Code-Generierung**: Erstellung neuer Funktionen, Methoden oder ganzer Module auf Basis von Kurzbeschreibungen oder natürlicher Sprache.
- **Testing und Qualitätssicherung**: Automatisierte 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, teilweise mit Bezug auf bestehende Anforderungen oder Projektdaten [Esp].

Rolle dieser Modelle für spezifische Aufgaben

- **Codegenerierung:** LLMs können auf Grundlage von Prompts oder bestehenden Code-Fragmente eigenständig funktionsfähigen Quellcode erzeugen. Dies umfasst Routineaufgaben (z. B. Boilerplate-Code) ebenso wie komplexere Algorithmen.
- **Testing:** Modelle wie GPT-4 sind in der Lage, automatisiert Tests zu erzeugen, Testdaten zu variieren und gängige Fehlerbilder zu erkennen.
- **Architekturvorschläge:** Moderne LLMs unterstützen zunehmend beim Entwurf und bei der Dokumentation von Softwarearchitekturen, indem sie z. B. Requirements in Design-Vorschläge oder Diagramme übersetzen [Esp, ND].

Herausforderungen und Grenzen Trotz des enormen Potenzials bestehen aktuelle Herausforderungen:

- **Halluzinationen und Fehleranfälligkeit**: Generative Modelle können syntaktisch korrekten, aber fachlich unpassenden oder sogar gefährlichen Code erzeugen.

- **Erklärbarkeit und Transparenz**: Die Nachvollziehbarkeit, wie ein Modell zu bestimmten Ergebnissen kommt, ist oft eingeschränkt [Esp, ND].
- **Domänenspezifisches Wissen**: Ohne Anpassung (Fine-Tuning) auf projektspezifische Daten sind die Modelle oft auf allgemeines Wissen beschränkt und berücksichtigen spezifische Anforderungen nur begrenzt.

Bezug zu den im Praxisteil genutzten Tools Die im Praxisteil eingesetzten Tools (*GitHub Copilot, Cursor, v0*) nutzen genau diese Algorithmen, um Entwickler:innen bei alltäglichen Entwicklungsaufgaben zu unterstützen. Sie liefern damit nicht nur klassische Code-Vervollständigungen, sondern wirken zunehmend als kollaborative Partner im gesamten Entwicklungsprozess – von Architektur über Implementierung bis 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 verstehen und daraus konkrete Vorschläge für Code, Tests oder Dokumentation generieren. Typisch ist ein *Prompt-Output-Paradigma*: Der oder die Entwickler*in gibt einen Prompt ein (z. B. eine Aufgabenbeschreibung oder einen Funktionsnamen), worauf das KI-Tool passenden Code vorschlägt. Besonders in modernen Entwicklungsumgebungen (IDEs) erscheinen diese Vorschläge direkt während des Tippens (Code Completion) oder als vollständige Funktionsblöcke [Ker, Wei].

2.2.2 Schnittstellen und Integration in Entwicklungsumgebungen

Die Integration generativer KI erfolgt überwiegend über IDE-Plugins (z. B. Visual Studio Code, JetBrains) oder über APIs. GitHub Copilot lässt sich als Erweiterung in gängigen Editoren installieren und unterstützt Entwickler*innen direkt im Arbeitsfluss. Zusätzlich existieren Chat-basierte Interaktionen, um komplexere Aufgaben wie Refactoring, Debugging oder Testautomatisierung zu erleichtern. Die nahtlose Einbindung in bestehende Entwicklungsumgebungen macht die Nutzung besonders praxisnah und niedrigschwellig [Ker, Shi, Wei].

2.2.3 Beispielhafte Workflows (Pair Programming mit Copilot)

Im Pair Programming mit GitHub Copilot formulieren Entwickler*innen Aufgaben als Prompt (z. B. “Implementiere eine Authentifizierung in React”). Copilot generiert dazu passenden Beispielcode, der übernommen oder angepasst werden kann. Die Entwickler*innen prüfen die Vorschläge und geben Feedback, indem sie unerwünschte Vorschläge ablehnen oder anpassen. Copilot kann während des gesamten Entwicklungsprozesses eingesetzt werden – etwa für Testautomatisierung, Refactoring oder Dokumentation. Studien zeigen, dass insbesondere Routinetätigkeiten dadurch erheblich beschleunigt werden [Ker, Wei, Shi].

2.2.4 Vorteile und Optimierungspotenziale

Zu den Vorteilen generativer KI-Tools zählen insbesondere:

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

Weitere Optimierungspotenziale ergeben sich durch die kontinuierliche Verbesserung der zugrundeliegenden KI-Modelle sowie die anpassbare Integration in unterschiedliche Projekte [Ker, Wei].

2.2.5 Grenzen und typische Fehlerquellen

Generative KI-Tools sind nicht fehlerfrei. Häufige Grenzen und Fehlerquellen sind:

- **Halluzinationen:** Die KI kann fehlerhaften oder unsicheren Code vorschlagen, insbesondere bei unpräzisen Prompts [Shi].
- **Bias und Kontextdefizite:** Die KI reproduziert möglicherweise Vorurteile oder ignoriert spezifische Projektregeln.
- **Sicherheitsrisiken:** Studien zeigen, dass Copilot mitunter unsicheren Code (z. B. SQL-Injection, Hardcoded Credentials) vorschlägt, sofern nicht explizit nach

sicheren Lösungen gefragt wird. Moderne Versionen reagieren allerdings auf gezielte Prompts und schlagen dann sichere Muster vor [Shi].

- **Übermäßiges Vertrauen:** Entwickler*innen übernehmen KI-Vorschläge manchmal ungeprüft. Daher sind Mechanismen zur kritischen Prüfung essenziell [Wei].

2.2.6 Designprinzipien für den produktiven und sicheren Einsatz

Für den erfolgreichen und sicheren Einsatz generativer KI-Tools gelten nach Weisz et al. (2024) folgende Designprinzipien:

- **Design for Mental Models:** Tools sollten so gestaltet sein, dass Nutzer*innen die Funktionsweise nachvollziehen können.
- **Design for Appropriate Trust & Reliance:** Mechanismen zur Förderung von Vertrauen und kritischer Prüfung sollten eingebaut werden (z. B. Hinweise auf Unsicherheiten, Feedbackmechanismen).
- **Design for Imperfection:** Nutzer*innen müssen aktiv auf mögliche Fehler hingewiesen und zur Korrektur befähigt werden [Wei].

Quellen:

- [Gey],[Ker],[Wei],[Marb],[Shi]

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.

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 zur automatischen Codegenerierung und Vervollständigung in Visual Studio Code.
- **Cursor:** KI-gestützte Entwicklungsumgebung mit besonderem Fokus auf Kontextintegration (Screenshots, Code, Fehlermeldungen) und dialogorientiertem Prompt Chaining.
- **Bolt.new:** Cloudbasierte Entwicklungsplattform mit direkter Anbindung an GitHub, automatisierter Fehlerbehebung und Multi-Plattform-Unterstützung (Mobile und Web).

Die Auswahl der 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 an ein junges, urbanes Publikum und zielt darauf ab, soziale Interaktionen rund um Veranstaltungen zu fördern.

Technologischer Stack und Architektur

- **Frontend:** Entwicklung mit React Native, TypeScript und Expo zur plattformübergreifenden Bereitstellung für iOS und Android.
- **Backend:** Nutzung von Firebase für Authentifizierung, Datenhaltung und Synchronisation.
- **Navigation:** Einsatz von `@react-navigation/native` und `expo-router` für ein modernes, tab-basiertes Navigationskonzept.

- **State-Management:** Eigene Context-Provider (`AuthProvider`, `EventsProvider`) für Authentifizierungs- und Eventdaten.
- **UI-Komponenten:** Verwendung von `@expo/vector-icons` und `lucide-react-native` zur Gestaltung eines ansprechenden User Interface.
- **Maps & Location:** Die Integration von `react-native-maps` und `expo-location` ermöglicht eine interaktive Kartenansicht, die sowohl den Nutzerstandort als auch Events auf einer Karte anzeigt.

Die App ist modular aufgebaut und umfasst drei zentrale Bereiche:

- **Explore-Screen:** Event-Feed nach Interessen und Standort
- **Map-Screen:** Interaktive Kartenansicht mit Event-Markern und Filterfunktion
- **Profil-Screen:** Persönliche Eventübersicht und Verwaltung

Während der 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 anschließenden Kapitel.

Struktur der Haupteinstiegskomponente (RootLayout)

Die zentrale Einstiegskomponente der App ist das `RootLayout`. Sie übernimmt das Laden benutzerdefinierter Fonts, die Einbindung von Authentifizierungs- und Events-Kontexten sowie die nahtlose Nutzerführung nach dem Login. Die Navigation wird dabei strikt vom Authentifizierungsstatus gesteuert, sodass nicht eingeloggte Nutzer:innen automatisch zur Login-Ansicht weitergeleitet werden. Dies wird mit Hilfe der Hooks `useAuth`, `useSegments` und `useRouter` umgesetzt.

3.2.2 Bestehende Funktionalitäten

Zu den bereits implementierten Basisfunktionen von Locals gehören:

- **Benutzerauthentifizierung:** 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:** Durch Nutzung von `react-native-safe-area-context` und Expo UI-Komponenten wird eine konsistente Darstellung auf unterschiedlichen Geräten gewährleistet.

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 demonstriert und analysiert.

3.3 Implementierung der interaktiven Kartenansicht mit KI-Unterstützung

3.3.1 Integration der KI-Tools

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. Ziel war die Umsetzung eines Map-Screens mit Event-Markern, Filterfunktion sowie Detailansicht – sämtliche Eventdaten stammen dabei aus dem zentralen EventsProvider.

Vor Beginn der Implementierung wurde Copilot aktiviert und die Entwicklungsumgebung vorbereitet (z. B. Installation von `react-native-maps` und `expo-location`). Die Aufgabenstellung für Copilot wurde jeweils als präziser Kommentar oder Docstring formuliert. Beispielsweise:

```
1 // Create a React Native component for a map with event markers and
      filter functionality
```

Oder als umfangreicher Prompt:

```
1 // Create a Map Screen with Event Markers and Filter in React Native.
      Requirements:
2 //
3 // - Use event data from the EventsProvider context
```

```
// - Show markers, enable filtering, display event details in modal/
    callout
// - Clean, modular, and maintainable code, with modern UI/UX
```

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 Schritt für Schritt 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 nach Rücksprache mit Copilot, durch Google-Suche oder manuelle Nacharbeit behoben.

Beispiel: Zur Erstellung des MapScreens wurde folgender Prompt verwendet (gekürzt):

```
// Create a React Native component called `MapScreen` that displays event
    markers on a map using event data from the `EventsProvider` context.
    Requirements:
//
// - Show all events as markers on a map
// - When a marker is tapped, show details
// - Add filter options above the map
// - Handle loading/error states appropriately
```

Die technische Umsetzung umfasste:

- Anzeige aller Events als Marker (mit Kategorie und Titel) auf der Karte (`react-native-maps`).
- Umsetzung einer Filterleiste zur Auswahl nach Kategorie und Datum.
- Darstellung von Event-Details beim Tippen auf einen Marker (Callout oder Modal).
- Responsives Layout und modernes UI-Design nach Vorgabe.
- Fehler- und Ladezustände wurden entsprechend behandelt.

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

Herausforderungen und Beobachtungen

Im Verlauf der Entwicklung zeigte Copilot verschiedene Stärken und Schwächen:

Stärken:

- Effiziente Generierung von Boilerplate-Code und wiederkehrenden Patterns (z. B. Hook-Nutzung, State-Management).
- Schnelle Vorschläge für UI-Komponenten und einfache Interaktionslogik.
- Erkennung von einfachen Fehlern und automatische Typanpassung nach Änderungen (z. B. bei Änderung der Datenstruktur für Kategorien).

Schwächen und typische Fehlerquellen:

- Teilweise fehlerhafte oder veraltete Import-Pfade (z. B. bei Kategorien).
- Missverständnisse bei nicht exakt spezifizierten Datenstrukturen (z. B. Umwandlung von Kategorien-Array von String zu Objekt wurde erst nach manuellem Eingriff korrekt erkannt).
- Filter-Logik für den “All”-Filter funktionierte zunächst nicht wie erwartet; Copilot schlug Anpassungen vor, die jedoch neue Fehler erzeugten (zwei “All”-Filter, Duplicate-Key-Warning).
- In mehreren Iterationen blieb die Korrektur von Spezialfällen (wie Filter-Probleme oder Typkonflikte) hinter den Erwartungen zurück, sodass letztlich manuelle Nachbesserung notwendig wurde.

Reflexion:

- Die Vorschläge waren bei Standardaufgaben meist brauchbar (*Subjektive Zufriedenheit: 4/5*), bei komplexeren State- oder Typ-Logiken aber oft unvollständig.
- Die Interaktion mit Copilot war intuitiv, erfordert jedoch genaue Prompts und ein grundsätzliches Verständnis für die Implementierung, um Fehlerquellen zu erkennen und zu korrigieren.
- Bei UI-Details oder individuelleren Anforderungen blieb Nacharbeit unerlässlich.

Fazit: Copilot ist ein sehr leistungsfähiges Assistenz-Tool, das Routinearbeiten und Standardaufgaben erheblich beschleunigt. Bei komplexeren oder individuelleren Anforderungen stößt es jedoch an Grenzen, sodass eine kritische Prüfung und manuelle Nacharbeit weiterhin unverzichtbar bleibt. 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). Cursor ermöglicht Prompt Chaining, die Verarbeitung von Screenshots als Referenz und einen dialogbasierten Entwicklungsprozess.

Zu Beginn wurden Screenshots des aktuellen App-Zustands sowie relevanter Komponenten (u. a. `_layout.tsx`, Event Provider) als Kontext bereitgestellt. Der Hauptprompt für den Einstieg lautete:

```
1 // Create a Map Screen with Event Markers and Filter in React Native.
      Requirements:
2 //
3 // - Use event data from the EventsProvider context
4 // - Show markers, enable filtering, display event details in modal/
      callout
5 // - Clean, modular, and maintainable code, with modern UI/UX
6 // - Use provided screenshots/layouts as visual reference
```

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 (Î1.24.3 statt Î1.18.0), und schlug proaktiv eine Anpassung auf die funktionierende Version vor.

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 (z. B. Kategorien als Objekte statt Strings) sehr präzise.
- 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

- **Versionierung und Abhängigkeiten:** Kompatibilitätsprobleme zwischen `react-native-maps` und Expo führten zu Fehlern, die erst nach mehreren Iterationen und Prompts gelöst wurden.
- **Automatische Code-Generierung:** Cursor wechselte in einem Schritt das Map-Framework (von `react-native-maps` auf `react-native-webview`), was unerwünscht war und manuell rückgängig gemacht wurde.
- **Filterfunktion:** Die Behandlung von Kategorie-Filtern (`All` vs. `all`) führte zu denselben Herausforderungen wie bei Copilot. Allerdings erkannte Cursor den Fehler nach gezieltem Prompt korrekt und schlug eine funktionierende Lösung vor.
- **Erkennung von Typfehlern:** Cursor reagierte auf TypeErrors (Objekte statt Strings in den Kategorien) konsistent und ergänzte die notwendigen Anpassungen selbstständig.

Reflexion:

- Die Entwicklung mit Cursor verlief insgesamt sehr zügig, da Kontextinformationen (Screenshots, Codeausschnitte) effektiv genutzt wurden.
- Die Vorschläge für komplexe UI- und Layout-Probleme waren oft präziser und anpassungsfähiger als bei Copilot.
- Der dialogische Ablauf mit Feedback-Loops und Prompt Chaining war besonders für iteratives Refactoring und das Lösen komplexerer Zusammenhänge hilfreich.
- Bei seltenen „KI-Fehlinterpretationen“ (z.B. Austausch ganzer Packages) 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

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. Ein dedizierter Branch (`bolt`) wurde verwendet. Das zugrundeliegende Sprachmodell war Claude 3.7 Sonnet (nicht direkt auswählbar).

Zunächst wurden Screenshots des aktuellen App-Zustands sowie zentrale Komponenten (u. a. `_layout.tsx`, Event Provider) als Kontext bereitgestellt. Die Aufgabenstellung für Bolt wurde als umfangreicher Prompt formuliert:

```
1  // Create a Map Screen with Event Markers and Filter in React Native.
2  // Create a React Native component called MapScreen that displays
       event markers on a map using event data from the EventsProvider
       context.
3  // Requirements:
4  //
5  // - Use the list of events from the EventsProvider context
6  // - Display all events as markers on a map
7  // - Add filter options above the map (by category, date, distance)
8  // - When a marker is tapped, show a callout or modal with event
       details
9  // - Modern mobile UI, modular and maintainable code
10 // - Use provided screenshots as visual reference
```

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`.

3. **Automatisiertes Debugging:** Terminalbefehle wie `npm run dev` wurden selbständig 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:

- Bolt war in der Lage, das Setup und viele Standard-Probleme selbständig 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 lassen sich deutliche Unterschiede und Gemeinsamkeiten in Bezug auf Funktionalität, Effizienz, Entwicklererlebnis und Ergebnisqualität feststellen.

Direkter Vergleich

- **Copilot** überzeugte besonders bei Standardaufgaben und bewährten Patterns in der Codegenerierung. Die Effizienzsteigerung bei Routinearbeiten ist erheblich, bei komplexeren Aufgaben oder individuellen Anforderungen blieb jedoch manuelle Nacharbeit nötig.

- **Cursor** ermöglichte durch den Kontextbezug (Screenshots, Code, Fehlermeldungen) und das dialogische Prompt Chaining eine besonders zielgerichtete und schnelle Entwicklung. Die Fehlerdiagnose und Lösungsvorschläge waren präziser als bei Copilot, allerdings erforderte auch Cursor für Spezialfälle aktive Begleitung durch die Entwickler:in.

- **Bolt** überzeugte mit seiner umfassenden Plattformintegration (z.B. GitHub, Web, App Store) und der Möglichkeit, Projekte direkt aus der Cloud-IDE zu initialisieren und zu deployen. Besonders im Web-Kontext zeigte Bolt große Stärken, stieß jedoch bei der mobilen Entwicklung und beim refactoring bestehender, größerer Projekte an Grenzen.

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 hochwertiger und passgenauer ist das Ergebnis. Unscharfe Prompts führen häufiger zu Fehlinterpretationen oder unbrauchbaren Vorschlägen.

- **Manuelle Kontrolle bleibt notwendig:** Auch bei hoher Automatisierung ist die Überprüfung aller KI-generierten Änderungen unerlässlich – insbesondere bei komplexen State- oder Typ-Logiken und individuellen Anforderungen.

- **Kontextnutzung und Feedback-Loops:** Tools wie Cursor, die Kontextinformationen (z. B. Screenshots, Fehlermeldungen) aktiv verarbeiten, bieten klare Vorteile bei komplexeren Aufgaben. Iteratives Prompt Chaining und direkte Rückmeldung sind Best Practice.

- **Tool-Auswahl nach Projekttyp:** Für „grüne Wiese“-Projekte, schnelle Prototypen und Web-Deployments sind Tools wie Bolt ideal. Für laufende oder größere Projekte empfiehlt sich der gezielte Einsatz von Copilot (bei Routineaufgaben) oder Cursor (bei komplexeren, dialogischen Arbeitsweisen).

- **Plattformkompatibilität beachten:** Unterschiedliche Plattformen (Web, Mobile) erfordern ggf. verschiedene Libraries oder Anpassungen – dies wird von den Tools unterschiedlich gut unterstützt.

Qualitative Bewertung: Zeiteffizienz, Codequalität und Wartbarkeit

Die qualitative Analyse der Entwicklung mit den drei generativen KI-Tools zeigt differenzierte Ergebnisse hinsichtlich Effizienz, Codequalität und Wartbarkeit:

- **Copilot** ermöglichte insbesondere bei Standardaufgaben eine spürbare Zeitersparnis. Routinemäßige Komponenten und einfache Logik wurden effizient und weitgehend fehlerfrei generiert. Die Codequalität war bei wiederkehrenden Patterns solide, bei komplexeren Strukturen und individueller Logik jedoch schwankend – hier war zusätzliche Nacharbeit erforderlich, um Wartbarkeit und Konsistenz zu gewährleisten.
- **Cursor** überzeugte durch schnelles Debugging und zielgerichtete Entwicklung, insbesondere bei der Integration von Kontextinformationen (Screenshots, Fehlermeldungen). Die Zeiteffizienz war bei komplexeren Aufgaben und bei Fehlerbehebung deutlich höher als bei Copilot. Die generierte Codequalität profitierte von der iterativen, dialogischen Zusammenarbeit, wodurch auch die Wartbarkeit des Endprodukts verbessert werden konnte.
- **Bolt** zeigte große Stärken in der Initialisierung neuer Projekte und bei Multi-Plattform-Support (Web, Mobile). Die Zeitersparnis war insbesondere im Setup und bei Standardfunktionalitäten signifikant. Im laufenden Betrieb und bei bestehenden Projekten traten jedoch häufiger Kompatibilitätsprobleme auf, die die Wartbarkeit und langfristige Codequalität beeinträchtigten und zusätzliche manuelle Eingriffe erforderten.

3.3.6 Zwischenfazit

Die praktische Demonstration hat zentrale Erkenntnisse für die weitere Analyse geliefert: Die Arbeit mit generativen KI-Tools ermöglicht eine signifikante Effizienzsteigerung bei der Umsetzung wiederkehrender Entwicklungsaufgaben und zeigt deutliche Potenziale im Bereich der Codequalität und Wartbarkeit. Gleichzeitig wurden spezifische Herausforderungen im Bereich der Fehlerbehebung, Tool-Integration und plattformübergreifenden Entwicklung sichtbar. Die im praktischen Teil gewonnenen Erfahrungen bilden die Grundlage für die weiterführende Bewertung der Chancen und Risiken generativer KI in der Softwareentwicklung, wie sie in den folgenden Kapiteln systematisch analysiert werden.

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.

> „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]

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.

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]).

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.

Im Blog von Fraunhofer IESE [Sie] wird betont, dass diese neuen Tools nicht nur als Autovervollständigung 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.

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.

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.

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 in der Softwareentwicklung ist mit zahlreichen technischen und organisatorischen Hürden verbunden. Nguyen-Duc et al. [ND] machen deutlich, dass viele Unternehmen Schwierigkeiten bei der Integration von GenAI-Tools in bestehende Entwicklungsprozesse haben. Ein zentrales Problem ist die fehlende Standardisierung von Schnittstellen, Datenformaten und Entwicklungsumgebungen, was die nahtlose Integration verschiedener Tools erschwert.

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.

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.

6 Wirtschaftliche und gesellschaftliche Auswirkungen

6.1 Veränderungen in Softwareunternehmen

- Auswirkungen auf Geschäftsmodelle und Prozesse
- Veränderungen in der Softwareentwicklung und im Projektmanagement
- Rolle von KI bei der Automatisierung von Softwareentwicklungsaufgaben

6.2 Auswirkungen auf den Arbeitsmarkt und Entwickler:innen-Rollen

- Verschiebung der gefragten Kompetenzen und Qualifikationen
- Neue Berufsbilder und veränderte Karrierewege
- Auswirkungen auf die Arbeitsplatzsicherheit und die Notwendigkeit der Weiterbildung

6.3 Zukunftsperspektiven und strategische Empfehlungen

- Notwendige Anpassungen für Unternehmen und Entwickler
- Möglichkeiten der Integration von KI in bestehende Entwicklungsprozesse
- Regulatorische und ethische Implikationen für eine nachhaltige KI-Nutzung

6.4 Kosten-Nutzen-Analyse von KI-gestützter Softwareentwicklung

- Analyse der wirtschaftlichen Effizienz und Kostenersparnis

- Vergleich der Investitionskosten und erwarteten Produktivitätsgewinne
- Langfristige wirtschaftliche Auswirkungen für Unternehmen und die Softwarebranche

7 Fazit und Ausblick

7.1 Erwartete Erkenntnisse

Es wird erwartet, dass KI-gestützte Softwareentwicklung nicht nur Effizienzsteigerungen ermöglicht, sondern auch die Arbeitsweise von Entwicklern nachhaltig verändert. Besonders relevant ist die Frage, inwieweit generative KI langfristig klassische Programmieraufgaben übernimmt oder ergänzt. Darüber hinaus soll analysiert werden, welche Herausforderungen in Bezug auf Sicherheit, Ethik und wirtschaftliche Auswirkungen entstehen.

7.2 Zusammenfassung der Erkenntnisse

7.3 Handlungsempfehlungen und Zukunftsperpektiven

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] Anantrasirichai, Nantheera; Zhang, Fan und Bull, David: Artificial Intelligence in Creative Industries: Advances Prior to 2025, URL http://arxiv.org/abs/2501.02725

[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

[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

[Hab] Habibi, Mahyar: Open Sourcing GPTs: Economics of Open Sourcing Advanced AI Models, URL http://arxiv.org/abs/2501.11581

[Has] Hassan, Ahmed E; Oliva, Gustavo A; Lin, Dayi und Chen, Boyuan: Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap

[Haz] Hazra, Sanchaita; Majumder, Bodhisattwa Prasad und Chakrabarty, Tuhin: AI Safety Should Prioritize the Future of Work, URL http://arxiv.org/abs/2504.13959

[Isl] Islam, Mohammad Rubyet und Sandborn, Peter: Multimodal Generative AI for Story Point Estimation in Software Development

[J] J, Bhuvana; Ranjan, Vivek und Bhaduariya, Nirmednra: Integration of AI in the Realm of Software Development, in: *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, URL https://ieeexplore.ieee.org/document/10465893

[Kar] Karhu, Katja; Kasurinen, Jussi und Smolander, Kari: Expectations vs Reality – A Secondary Study on AI Adoption in Software Testing, URL http://arxiv.org/abs/2504.04921

[Ker] Kerr, Kedasha: GitHub for Beginners: Building a React App with GitHub Copilot - The GitHub Blog

[Kha] Khan, Abidullah; Shokrizadeh, Atefeh und Cheng, Jinghui: Beyond Automation: How UI/UX Designers Perceive AI as a Creative Partner in the Divergent Thinking Stages, S. 1–12, URL http://arxiv.org/abs/2501.18778

[Lee] Lee, Kyungho: Towards a Working Definition of Designing Generative User Interfaces, URL http://arxiv.org/abs/2505.15049

[Mara] Marguerit, David: Augmenting or Automating Labor? The Effect of AI Development on New Work, Employment, and Wages, URL http://arxiv.org/abs/2503.19159

[Marb] Martinović, Boris und Rozić, Robert: Impact of AI Tools on Software Development Code Quality, in: Tomislav Volarić; Boris Crnokić und Daniel Vasić (Herausgeber) *Digital Transformation in Education and Artificial Intelligence Application*, Springer Nature Switzerland, URL https://link.springer.com/chapter/10.1007/978-3-031-62058-4_15

[Mat] Matsumoto, Kenichi: Conceptual Framework for Next-Generation Software Ecosystems, in: *2021 IEEE/ACIS 22nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, URL https://ieeexplore.ieee.org/document/9705010

[McN] McNamara, Kevin J. und Marpu, Rhea Pritham: Exponential Shift: Humans Adapt to AI Economies, URL http://arxiv.org/abs/2504.08855

[Min] Minikiewicz, Arlene: The Impact of Generative AI on Software Engineering Activities

[ND] Nguyen-Duc, Anh; Cabrero-Daniel, Beatriz; Przybylek, Adam; Arora, Chetan; Khanna, Dron; Herda, Tomas; Rafiq, Usman; Melegati, Jorge; Guerra, Eduardo; Kemell, Kai-Kristian; Saari, Mika; Zhang, Zheying; Le, Huy; Quan, Tho und Abrahamsson, Pekka: Generative Artificial Intelligence for Software Engineering – A Research Agenda, URL https://arxiv.org/abs/2310.18648, _eprint: 2310.18648

[noa] Verordnung (EU) 2024/1689 des Europäischen Parlaments und des Rates vom 13. Juni 2024 zur Festlegung harmonisierter Vorschriften für künstliche Intelligenz und zur Änderung der Verordnungen (EG) Nr. 300/2008, (EU) Nr. 167/2013, (EU) Nr. 168/2013, (EU) 2018/858, (EU) 2018/1139 und (EU) 2019/2144 sowie der Richtlinien 2014/90/EU, (EU) 2016/797 und (EU) 2020/1828 (Verordnung über künstliche Intelligenz)Text von Bedeutung für den EWR.

[Ric] Richards, Jonan und Wessel, Mairieli: Bridging HCI and AI Research for the Evaluation of Conversational SE Assistants, URL http://arxiv.org/abs/2502.07956

[Rog] Rogachev, Daniel: My Experience with GitHub Copilot Agent: Developing a React Native Application | LinkedIn

[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, Maliheh: Human-AI Experience in Integrated Development Environments: A Systematic Literature Review, URL http://arxiv.org/abs/2503.06195

[Shi] Shi, Yong; Sakib, Nazmus; Shahriar, Hossain; Lo, Dan; Chi, Hongmei und Qian, Kai: AI-Assisted Security: A Step towards Reimagining Software Development for a Safer Future, in: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, URL https://ieeexplore.ieee.org/document/10196879

[Sie] Siebert, Dr. J. und Jedlitschka, Dr. Andreas: Generative KI im Software Engineering: Szenarien und künftige Entwicklungen, URL https://www.iese.fraunhofer.de/blog/generative-ki-softwareentwicklung

[Sif] Sifi, Adlene: How does generative AI impact Developer Experience? URL https://devblogs.microsoft.com/premier-developer/how-does-generative-ai-impact-developer-experience/

[Son] Song, Fangchen; Agarwal, Ashish und Wen, Wen: The Impact of Generative AI on Collaborative Open-Source Software Development: Evidence from GitHub Copilot. URL https://www.ssrn.com/abstract=4856935, publisher: Elsevier BV

[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

Tabellenverzeichnis

Listings

.

A Anhang 1

B Anhang 2