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

26. Juni 2025

Referent: Prof. Dr. Tim Mustermann

Korreferent: Prof. Dr. Max Häuser

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 26. 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 [Sie24].

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.24]) . 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

- Überblick über generative KI-Modelle
- Relevante Algorithmen und Methoden in der Softwareentwicklung

2.1.1 Beispiele für generative KI-Tools in der Praxis

2.1.2 Wichtige Algorithmen und Modelle in der Softwareentwicklung

2.2 Generative KI-Tools: Funktion und Anwendung

- Beispiele für generative KI-Tools in der Praxis
- Einsatzmöglichkeiten und Optimierungspotenziale

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 [Don24, Cou24, S.24, Bra24, Sie24]. 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.“ [Don24, 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.24]. 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 Siegeszug generativer KI hat in den letzten Jahren eine Vielzahl innovativer Werkzeuge und Methoden in die Softwareentwicklung eingeführt. Insbesondere die Entstehung von KI-gestützten Entwicklungsumgebungen und Coding-Assistenten wie GitHub Copilot, Cursor oder Bolt.new hat dazu geführt, dass Entwickler:innen in ihrem Workflow neue Arbeitsweisen adaptieren müssen [Don24, Wei24, Sie24].

Im praktischen Teil dieser Arbeit (vgl. Kapitel 3) zeigte sich deutlich, wie unterschiedlich diese Tools in der Entwicklungspraxis agieren und welche neuen Möglichkeiten sie eröffnen. Während Copilot insbesondere für die schnelle Generierung von Standard-Code, Boilerplate und UI-Elementen eingesetzt werden konnte, überzeugte Cursor durch seine Fähigkeit, Kontextinformationen wie Screenshots oder Fehlermeldungen direkt in die Codegenerierung und Problemlösung zu integrieren. Bolt.new wiederum ermöglichte eine weitgehend automatisierte Initialisierung und das cloudbasierte Setup von Projekten, inklusive direkter Web-Deployments.

Diese neuen Werkzeuge führen zu einer Verschiebung der klassischen Rollenverteilung und eröffnen neue Methoden im Entwicklungsprozess. Die Literatur beschreibt, dass insbesondere die Integration von Large Language Models (LLMs) und Retrieval-Augmented Generation (RAG) zu einer höheren Anpassungsfähigkeit und Kontextsensitivität führt [**?**, **?**]. Methoden wie Prompt Chaining, dialogische Interaktion mit der KI oder die Nutzung von Kontextdaten in Echtzeit gehören zunehmend zum Alltag in modernen Entwicklerteams [**?**].

> „Tools like Copilot and Cursor AI have introduced dialog-driven, context-aware development workflows, in which code suggestions, debugging, and refactoring are conducted interactively, often utilizing screenshots or project artifacts as additional context.“ [**?**, S. 10]

Ein weiteres zentrales Ergebnis der praktischen Demonstration ist die Erkenntnis, dass die Qualität der Ergebnisse maßgeblich von der Fähigkeit zur Integration und Steuerung der KI im Entwicklungsprozess abhängt. Besonders Cursor konnte durch die Nutzung von Prompt Chaining und der aktiven Einbindung von User-Feedback die Entwicklungsgeschwindigkeit und Problemlösungskompetenz deutlich steigern (vgl. Kapitel 3). Die Erfahrung zeigt zudem, dass die Wahl des passenden Tools und der dazugehörigen Methode stark vom Projekttyp, den Teamstrukturen und den angestrebten Zielen abhängt.

Insgesamt lässt sich festhalten, dass generative KI nicht nur neue Werkzeuge, sondern auch grundlegend neue Methoden und Arbeitsweisen in die Softwareentwicklung gebracht hat, die zukünftig weiter an Bedeutung gewinnen werden.

5 Herausforderungen durch KI in der Softwareentwicklung

Trotz der vielversprechenden Möglichkeiten von KI-gestützten Entwicklungsmethoden existieren Herausforderungen, die nicht vernachlässigt werden dürfen. Besonders Datenschutz- und Sicherheitsaspekte spielen eine zentrale Rolle, ebenso wie ethische Fragen zur Fairness und Transparenz von KI-Modellen. In diesem Kapitel werden die wesentlichen Problembereiche diskutiert, die mit der zunehmenden Integration von KI in die Softwareentwicklung verbunden sind.

5.1 Sicherheits- und Datenschutzaspekte

5.1.1 Sicherheitsrisiken durch generative Modelle

5.2 Ethische und soziale Implikationen

5.2.1 Ethische Konflikte und Bias in KI-Systemen

5.2.2 Langfristige Auswirkungen auf Entwickler:innen-Rollen

5.2.3 Technische und organisatorische Hürden bei der Einführung von KI

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.24] A., Downie und M., Finio: KI in der Softwareentwicklung (2024), URL https://www.ibm.com/de-de/think/topics/ai-in-software-development, iBM analysiert den Einfluss von KI auf die Softwareentwicklung und hebt hervor, wie KI-Technologien die Produktivität, Genauigkeit und Innovation steigern können. Der Artikel betont die Rolle von generativer KI und großen Sprachmodellen bei der Optimierung des Entwicklungszyklus.[Zugegriffen: 16. Januar 2025]

[Bra24] Braun, A. M.: KI in der Softwareentwicklung: Wie effektiv codet KI wirklich? (2024), URL https://www.computerwoche.de/article/2832991/wie-effektiv-codet-ki-wirklich.html, dieser Artikel untersucht die Effizienz von KI in der Softwareentwicklung und diskutiert, inwieweit KI-gestützte Tools den Programmierprozess unterstützen und verbessern können.[Zugegriffen: 18. Januar 2025]

[Cou24] Coutinho, Mariana; Marques, Lorena; Santos, Anderson; Dahia, Marcio; Franca, Cesar und de Souza Santos, Ronnie: The Role of Generative AI in Software Development Productivity: A Pilot Case Study (2024), URL https://arxiv.org/abs/2406.00560, diese Fallstudie untersucht den Einfluss generativer KI auf die Produktivität in der Softwareentwicklung und liefert erste empirische Erkenntnisse.

[Cui24] 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 (2024), URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4945566, die Autoren präsentieren Ergebnisse aus drei Feldexperimenten, die den Einfluss generativer KI auf die Arbeit von hochqualifizierten Softwareentwicklern untersuchen.

[Don24] 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, dieses Review bietet einen umfassenden Überblick über aktuelle generative KI-Tools in der Anwendungsentwicklung und

diskutiert deren Technologien und Praktiken.

[J23] 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, die Studie analysiert die Integration von KI in die Softwareentwicklung und identifiziert Vorteile, Herausforderungen und Best Practices für Entwickler und Organisationen.

[Mar24] 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, Cham, URL https://link.springer.com/chapter/10.1007/978-3-031-62058-4_15, die Autoren analysieren die Auswirkungen von KI-Tools auf die Codequalität in der Softwareentwicklung und identifizieren sowohl Vorteile als auch potenzielle Risiken.

[Mat21] 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, dieses Papier präsentiert ein konzeptionelles Framework für zukünftige Software-Ökosysteme und untersucht, wie KI-Technologien in diese integriert werden können, um Effizienz und Innovation zu fördern.

[ND23] 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 (2023), URL https://arxiv.org/abs/2310.18648, dieses Papier skizziert eine Forschungsagenda für den Einsatz generativer KI im Software Engineering und identifiziert offene Forschungsfragen in verschiedenen Bereichen.

[S.24] S., Sulabh: The Future of Coding is Here – How AI is Reshaping Software Development (2024), URL https://www.deloitte.com/uk/en/Industries/technology/blogs/2024/the-future-of-coding-is-here-how-ai-is-reshaping-software-development.html, deloitte untersucht, wie KI die Softwareentwicklung transformiert, und diskutiert die Auswirkungen auf Entwickler, Prozesse und die gesamte Branche. Der Artikel betont die Bedeutung von KI für die zukünftige Gestaltung der Softwareentwicklung.[Zugegriffen: 15. Januar 2025]

[Sch24] Schmitt, Anuschka; Gajos, Krzysztof Z. und Mokryn, Osnat: Generative AI in the Software Engineering Domain: Tensions of Occupational Identity

and Patterns of Identity Protection (2024), URL https://arxiv.org/abs/2410.03571, die Studie untersucht, wie generative KI die berufliche Identität von Softwareingenieuren beeinflusst und welche Strategien zum Schutz dieser Identität angewendet werden.

[Shi23] 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, die Autoren diskutieren, wie KI-gestützte Sicherheitslösungen die Softwareentwicklung sicherer gestalten können, indem sie Bedrohungen frühzeitig erkennen und proaktive Maßnahmen ermöglichen.

[Sie24] Siebert, Dr. J. und Jedlitschka, Dr. Andreas: Generative KI im Software Engineering: Szenarien und künftige Entwicklungen (2024), URL https://www.iese.fraunhofer.de/blog/generative-ki-softwareentwicklung, das Fraunhofer IESE beleuchtet verschiedene Anwendungsszenarien für generative KI im Software Engineering und diskutiert zukünftige Entwicklungen sowie potenzielle Herausforderungen in diesem Bereich.[Zugegriffen: 15. Januar 2025]

[Wan18] 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, die Autoren erforschen, wie KI-Techniken die automatisierte Wiederverwendung und das Design von Software unterstützen können, um Entwicklungszeiten zu verkürzen und die Qualität zu verbessern.

[Wei24] 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, in diesem Papier werden sechs Designprinzipien für generative KI-Anwendungen vorgestellt, die einzigartige Herausforderungen adressieren und Empfehlungen für die Gestaltung effektiver und sicherer Nutzererfahrungen bieten.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

.

A Anhang 1

B Anhang 2