Paech, Andreas

Article — Published Version

Objektorientierte Softwareentwicklung mit Generativer KI – Experimentelle Case Study zu Prompting Strategien mit GPT-4

HMD Praxis der Wirtschaftsinformatik

Provided in Cooperation with:
Springer Nature

Suggested Citation: Paech, Andreas (2025) : Objektorientierte Softwareentwicklung mit Generativer KI – Experimentelle Case Study zu Prompting Strategien mit GPT-4, HMD Praxis der Wirtschaftsinformatik, ISSN 2198-2775, Springer Fachmedien Wiesbaden GmbH, Wiesbaden, Vol. 62, Iss. 5, pp. 1231-1245, https://doi.org/10.1365/s40702-025-01147-x

This Version is available at:
https://hdl.handle.net/10419/330715

HMD Praxis der Wirtschaftsinformatik (2025) 62:1231–1245
https://doi.org/10.1365/s40702-025-01147-x

SPEKTRUM

Objektorientierte Softwareentwicklung mit Generativer KI – Experimentelle Case Study zu Prompting Strategien mit GPT-4

Andreas Paech

Eingegangen: 12. Dezember 2023 / Angenommen: 14. Januar 2025 / Online publiziert: 20. Februar 2025

Zusammenfassung Diese Studie untersucht die Anwendung und Effektivität von GPT-4 im Bereich der objektorientierten Softwareentwicklung zur Generierung von funktionalem und qualitativ hochwertigem Code. In den experimentellen Fallstudien wurde GPT-4 mit zunehmend komplexen Programmieraufgaben betraut. Der erste Fall, ein auf Java basierendes Studentenverwaltungssystem, demonstrierte die Fähigkeit von GPT-4, strukturierten, funktionalen Code als Reaktion auf spezifische Prompts zu generieren. Es zeigte jedoch auch die Notwendigkeit für präzise, kontextreiche Prompts auf, um wartbare und qualitativ hochwertige Softwarelösungen zu erstellen. Im zweiten Fall, das Spiel „Snake“, produzierte GPT-4 anfangs allgemeine Ansätze und Pseudocode und benötigte iteratives Prompting und Fehleranalyse, um ein funktionierendes Spiel zu entwickeln. Dieser Fall unterstrich die Grenzen von GPT-4 bei der Bewältigung komplexer Programmieraufgaben ohne detaillierte Anleitung. Die Ergebnisse dieser Studien deuten darauf hin, dass GPT-4 ein wertvolles Werkzeug in der Softwareentwicklung sein kann, wenn Prompting-Strategien mit einem tiefen Verständnis der Prinzipien der Softwareentwicklung kombiniert werden. Jedoch wird die Notwendigkeit menschlicher Expertise bei kritischen Entscheidungen und komplexen Designaufgaben hervorgehoben. Das Papier schließt mit einer Diskussion über die Integration von generativer KI in Techniken und Methoden des Software-Engineerings und die ethischen Implikationen von Verzerrungen, die in den Antworten von GPT-4 beobachtet wurden. Diese Studie trägt zum Verständnis des Potenzials und der Grenzen generativer KI in der Softwareentwicklung bei und unterstreicht die anhaltende Notwendigkeit menschlicher Aufsicht und Expertise in diesem Bereich.

✉ Andreas Paech
Fakultät für Wirtschafts- und Sozialwissenschaften, Universität Hamburg, Hamburg, Deutschland
E-Mail: andreas.paech@uni-hamburg.de

Schlüsselwörter Prompt Engineering · Generative KI · GPT-4 · Code Generation · Objektorientierte Programmierung · Softwareentwicklung · Case Study

Object-Oriented Software Development with Generative AI — Experimental Case Study for Prompting Strategies with GPT-4

Abstract This paper presents an experimental case study on the application of GPT-4 in the domain of object-oriented software development. The study aims to assess the effectiveness of GPT-4 in generating functional and high-quality code, exploring the role of prompt engineering in guiding the model to achieve specific software development objectives. In the experimental case studies, GPT-4 was tasked with progressively complex programming challenges. The first case, a Java-based student management system, demonstrated GPT-4's capacity to generate structured, functional code in response to specific prompts. However, it also revealed the need for precise, context-rich prompts to create maintainable and high-quality software solutions. The second case, the "Snake" game, presented a more complex scenario. GPT-4 initially produced general approaches and pseudocode, requiring iterative prompting and error analysis to develop a functional game. This case highlighted the limitations of GPT-4 in handling complex programming tasks without detailed guidance. The results of these studies suggest that GPT-4 can be a valuable tool in software development when combined with thoughtful prompting strategies and a deep understanding of software development principles. However, the necessity of human expertise in critical decision-making and complex design tasks is underscored. The limitations of GPT-4 in automated testing and its current inability to independently handle complex software projects are also discussed. The paper concludes with a discussion on the integration of generative AI in software engineering methodologies and the ethical implications of biases observed in GPT-4's responses. This study contributes to understanding the potential and limitations of generative AI in software development and highlights the ongoing need for human oversight and expertise in the field.

Keywords Prompt engineering · Generative AI · GPT-4 · Code generation · Object-oriented programming · Software engineering · Case study

1 Einleitung

Mit der Veröffentlichung von ChatGPT im November 2022 und den darauf folgenden Weiterentwicklungen wie GPT-3.5 Turbo, GPT-4 und o1 hat sich das Feld der generativen Sprachmodelle (GSM) rasant weiterentwickelt. Diese Entwicklung markiert eine disruptive Innovation in der Landschaft der Künstlichen Intelligenz (Teubner et al. 2023). Der Kern dieser Studie fokussiert sich auf die Effektivität von GPT-4 in der objektorientierten Softwareentwicklung und untersucht, inwieweit diese fortschrittlichen Modelle in der Lage sind, qualitativ hochwertigen und funktionalen Code zu generieren. Durch die Automatisierung von Programmier-

aufgaben und die Unterstützung bei der Fehlerbehebung können diese Modelle zu signifikanten Effizienzsteigerungen für IT-Professionals führen.

Es wurden dafür zwei Case Studies konzipiert: die Entwicklung einer textbasierten Studierendenverwaltung in Java und die Implementierung des Spiels „Snake“ mit Web-Technologien. Diese Projekte wurden nicht nur aufgrund ihrer unterschiedlichen technologischen Anforderungen ausgewählt, sondern auch, weil sie repräsentative Beispiele für Aufgaben darstellen, die typischerweise in der Ausbildung von (Wirtschafts-)Informatiker:innen behandelt werden. Diese Auswahl ermöglicht es, die Anwendbarkeit und Effektivität von GPT-4 in einem breiten Spektrum von Programmierkontexten zu untersuchen. Die beiden Cases finden zu verschiedenen Zeitpunkten der Entwicklung von GPT-4 statt, wodurch die kontinuierlichen Verbesserungen des Modells berücksichtigt wurden.

Die Forschungsfragen, die diese Studie leiten, sind:

Forschungsfrage 1 Wie effektiv ist der Einsatz von GPT-4 in der objektorientierten Softwareentwicklung?

Forschungsfrage 2 Welche Rolle spielt das Prompting bei der Steuerung von GPT-4 in der Softwareentwicklung?

Im Folgenden werden zunächst die theoretischen Grundlagen erläutert, bevor die Daten, Methode und Ergebnisse vorgestellt werden. Die Studie schließt mit einer Diskussion zur Software-Qualität, die Integration von Generativer Künstlicher Intelligenz in die Vorgehensmodelle in der Softwareentwicklung sowie der Beobachtung von Biased Verhalten durch GPT-4.

2 Theoretische Grundlagen

2.1 Objektorientierte Softwareentwicklung

Die Objektorientierung konzentriert sich auf die Strukturierung von Softwaresystemen in Form von Objekten. Jedes Objekt ist eine Instanz einer Klasse, die spezifische Daten (Attribute) und Verhaltensweisen (Methoden) enthält. Die Objektorientierung fördert Konzepte wie Vererbung[1], Polymorphismus[2], Kapselung[3] und Modularisie-

[1] *Vererbung* ermöglicht es, eine neue Klasse (auch Unter-, Kind- oder abgeleitete Klasse) von einer bestehenden Klasse (Super- oder Basisklasse) abzuleiten. Die Subklasse „erbt“ dabei Eigenschaften (Attribute) und Fähigkeiten (Methoden) der Basisklasse und kann diese erweitern oder anpassen (z. B. durch Überschreiben).

[2] *Polymorphismus* bezeichnet die Fähigkeit eines Objektes, in verschiedenen Aufrufkontexten unterschiedliche Formen anzunehmen. Ein einheitlicher Methodenaufruf kann damit für verschiedene Klassenvarianten verwendet werden, sodass dieselbe Methode je nach konkretem Objekttyp eine abweichende Implementierung besitzt.

[3] *Kapselung* verbirgt die internen Implementierungsdetails, indem Daten (Attribute) und Funktionen (Methoden) in einer Klasse zusammengefasst werden. Dadurch stehen nach außen nur definierte Zugriffspunkte zur Verfügung und Änderungen im Inneren der Klasse haben keine direkten Auswirkungen auf andere Codebereiche.

rung[4] was zu einer besseren Wartbarkeit und Erweiterbarkeit des Codes führt. Das **Model-View-Controller** (MVC) Design Pattern ist weit verbreitet bei einfachen Softwaresystemen. Das MVC-Pattern teilt eine Anwendung in drei Hauptbereiche auf: das Modell (für die Datenlogik), die Ansicht (die Benutzerschnittstelle) und den Controller (der die Interaktion zwischen Modell und Ansicht steuert). Darüber hinaus sind SOLID-Prinzipien wesentlich für Clean Code in der objektorientierten Softwareentwicklung. Sie umfassen **Single Responsibility** (jeder Programmbaustein sollte nur eine Verantwortung haben), **Open/Closed** (Klassen sollten für Erweiterung offen, aber für Modifikation geschlossen sein), **Liskov-Substitution** (Objekte einer Unterklasse sollten die Stelle ihrer Oberklasse einnehmen können), **Interface Segregation** (größere Interfaces sollten in kleinere, spezifischere aufgeteilt werden) und **Dependency Inversion** (Module höherer Ebenen sollten nicht von Modulen niedrigerer Ebenen abhängen). Clean Code fokussiert auf die Lesbarkeit und Einfachheit des Codes, indem er leicht verständliche Konventionen und klare Strukturen fördert, was zu besserer Wartbarkeit und Effizienz im Entwicklungsprozess führt.

2.2 Generative Sprachmodelle

Im Bereich der generativen Sprachmodelle verbinden sich die Konzepte neuronale Netze, Deep Learning und Large Language Models (LLMs) zu einem kohärenten System, das die Erzeugung von menschenähnlichen Texten ermöglicht. Neuronale Netze sind dabei ein Teilgebiet des maschinellen Lernens, bei dem mehrere Schichten aus vernetzten Knoten (oft als „Neuronen“ bezeichnet) zum Einsatz kommen, deren Struktur einem menschlichen Gehirn ähnelt. Deep Learning ist eine fortgeschrittene Form des maschinellen Lernens und nutzt mehrschichtige neuronale Netze, um komplexe Muster in Daten zu lernen und zu interpretieren. Diese Methodik ermöglicht es, dass die Modelle kontinuierlich aus einer enormen Menge an Textdaten lernen, wodurch sie in der Lage sind, Sprache in einer Tiefe zu verstehen, die frühere Technologien nicht erreichen konnten. Der Textkorpus zum Training von GPT-4 umfasst ca. 570 GB (Teubner et al. 2023).

LLMs wie GPT (Generative Pre-trained Transformer) repräsentieren die zur Zeit letzte Entwicklungsstufe. Sie sind auf die Verarbeitung und Generierung von Text spezialisiert und nutzen die Transformer-Architektur (Vaswani et al. 2017), eine innovative Architektur in neuronalen Netzen, die auf Self-Attention-Mechanismen basiert. Diese Mechanismen erlauben dem Modell, die Bedeutung von Wörtern im Kontext eines ganzen Satzes oder Absatzes zu verstehen, was zu einer präzisen und kohärenten Texterzeugung führt. Das Zusammenwirken dieser drei Konzepte – neuronale Netze, Deep Learning und LLMs mit Transformer-Architektur – bildet das Fundament für die fortschrittlichen Fähigkeiten der heutigen generativen Sprachmodelle wie ChatGPT, Bard/Gemini und Claude.

[4] *Modularisierung* bezeichnet das Aufteilen eines Softwaresystems in klar abgegrenzte, eigenständige Bausteine (Module), die über definierte Schnittstellen miteinander kommunizieren.

2.3 Prompt Engineering

Prompt Engineering ist ein Prozess, der in der Interaktion mit KI-basierten Sprachmodellen eingesetzt wird, um spezifische, effektive und präzise Antworten zu generieren. Es beinhaltet die sorgfältige Gestaltung von Eingabeaufforderungen (Prompts), die das Verhalten und die Antworten eines KI-Modells steuern. Dieser Prozess umfasst die Auswahl der richtigen Worte, Formulierungen und Kontextinformationen, um das KI-Modell zu leiten und die gewünschten Ergebnisse zu erzielen. Prompt Engineering ist entscheidend, da die Qualität und Spezifität des Prompts direkt die Relevanz und Genauigkeit der Antworten des Modells beeinflussen (Radford et al. 2018).

Iteratives Prompting, eine Kernstrategie, umfasst verschiedene Ansätze wie Zero-Shot, Few-Shot und gegebenenfalls Finetuning (Swaylu 2023). Beim **Zero-Shot-Prompting** (Kojima et al. 2022) wird das Modell ohne vorherige spezifische Beispiele oder Trainingsdaten auf eine Aufgabe angesetzt, wohingegen **Few-Shot-Prompting** einige wenige Beispiele zur Orientierung nutzt. **Finetuning**, weiterführendes Training für eine klar definierte Aufgabe, kann eingesetzt werden, um das Modell, unabhängig von den Prompts, mit Blick auf spezifische Anforderungen und Kontexte hin anzupassen (Swaylu 2023).

Eine weitere wichtige Technik ist die **Chain-of-Thought** (Wei et al. 2022; Zhang et al. 2022). Hierbei wird das Modell durch eine schrittweise Argumentation oder „Reasoning“ zu einer Lösung geführt, wobei oft ein Prozess des schrittweisen Denkens angewandt wird. Dieses kann auch durch die einfache Formulierung „step-by-step“ im Prompt getriggert werden. Diese Methode hilft, komplexe Probleme in kleinere, handhabbare Teile zu zerlegen und ermöglicht dem Modell, seine Gedankengänge transparent darzulegen und in der Regel zu einer besseren Lösung zu kommen (Mendes 2023).

Die **Separation of Concerns** ist ebenfalls zentral im Prompt- wie im Software-Engineering. Sie beinhaltet das gezielte Trennen verschiedener Aspekte einer Anfrage, um Klarheit und Fokus zu gewährleisten (Mendes 2023; Selvaraj 2023; Swaylu 2023; Yang 2023). Dies umfasst die präzise Formulierung von Instruktionen, die Verwendung von Trennzeichen zur Strukturierung von Prompts (Selvaraj 2023) und die bewusste Auswahl der Sprache, um die gewünschte Detailtiefe und das Format der Antworten zu steuern. Domain-spezifische Terminologie, sogenannte Leading Words, werden für das Nudging des Modells in eine bestimmte Richtung eingesetzt (Swaylu 2023).

Ein weiterer Aspekt ist das Management von Kontext und Daten. Dies beinhaltet das Einbringen relevanter Informationen und Hintergründe in die Prompts, um dem Modell eine fundierte Basis für seine Antworten zu geben (Mendes 2023; Selvaraj 2023; Swaylu 2023; Yang 2023), sogenanntes **in-context learning** (Wei et al. 2022). Personas und fortgeschrittene Rollenmodelle, wie das „Act as ...“-Schema, ermöglichen es, das Modell in eine spezifische Rolle oder Perspektive zu versetzen (Mendes 2023; Swaylu 2023; Yang 2023). Schließlich sind sowohl das Token-Management als auch die Modell-Temperatur entscheidend, um einerseits die Länge der generierten Antworten (z. B. durch Stoppwörter) zu steuern und andererseits den Grad der Zufälligkeit („Kreativität“) bei der Textproduktion zu regeln (Mendes 2023). Das

Token-Management ermöglicht eine gezielte Begrenzung und Anpassung des Ausgabeformats, während eine höhere Modell-Temperatur zwar kreativere, aber potenziell weniger präzise Resultate liefert. Eine niedrigere Temperatur führt entsprechend zu stabileren, jedoch weniger variantenreichen Antworten. Beide Stellgrößen beeinflussen somit maßgeblich die Effizienz und Relevanz der generierten Texte (Mendes 2023).

3 Daten und Methode

3.1 Experimentelle Case Study

Die Studie ist eine experimentelle Case-Study, um eine detaillierte Untersuchung der Anwendung von GPT-4 in der objektorientierten Softwareentwicklung vorzunehmen. GPT-4 wurden zwei Programmieraufgaben mit steigender Komplexität und unterschiedlichen Technologien gestellt, die es zu lösen hatte. Die Prompts beinhalten zunächst wenige Techniken des Prompt-Engineerings, um das Standardverhalten von GPT-4 explorativ untersuchen zu können.

Diese Methode wurde gewählt, um sowohl die Effektivität von GPT-4 in der Entwicklung realer Anwendungen als auch die Auswirkungen unterschiedlicher Prompts in einem kontrollierten, aber praxisnahen Umfeld zu bewerten. Um die dynamischen Weiterentwicklungen des GPT-4-Modells zu berücksichtigen, wurden zwei zeitlich versetzte Case Studies gewählt. Die erste, eine textbasierte Studierendenverwaltung in Java, fand in der Kalenderwoche 25 (2023) statt und nutzte GPT-4 mit einem Kontextfenster von 16k Tokens. Die zweite Case Study, die Implementierung des Spiels „Snake“ mit Web-Technologien, wurde in Kalenderwoche 48 durchgeführt und nutzte ein erweitertes Kontextfenster von 128k Tokens. Dies ermöglichte es, die Anpassungsfähigkeit und Leistung des Modells bei komplexeren und technisch anspruchsvolleren Aufgaben zu analysieren. Durch diese zeitliche Anordnung und die Berücksichtigung der Modell-Updates konnte ein umfassendes Bild von den Fähigkeiten und Grenzen von GPT-4 in verschiedenen Softwareentwicklungskontexten gewonnen werden.

Der erste Case, die Studierendenverwaltung in Java, ermöglicht es, die Fähigkeit von GPT-4 zur Umsetzung von objektorientierten Designmustern zu testen. Im Vergleich zu Web-Technologien ist die textbasierte Benutzeroberfläche weniger komplex, was eine differenzierte Bewertung der Backend-Programmierungsfähigkeiten von GPT-4 ermöglicht. Das zweite Projekt, das Spiel Snake, implementiert mit Web-Technologien, deckt den Bereich der Frontend-Entwicklung ab. Die Verwendung von HTML, CSS und JavaScript erfordert ein tiefes Verständnis der Interaktion zwischen diesen Technologien. Besonders relevant in diesem Kontext ist die Entwicklung von interaktiven und benutzerfreundlichen Frontend-Anwendungen. Das Projekt stellt GPT-4 vor die Herausforderung, ein dynamisches Spiel zu entwickeln, was sowohl die Handhabung komplexer UI/UX-Anforderungen als auch das Event-Handling und die Interaktivität betrifft.

3.2 Case 1: Textbasierte Studierendenverwaltung in Java

Es sollen die grundlegenden Funktionen einer Studierendenverwaltung, wie das Hinzufügen und Anzeigen von Studierenden, implementiert werden. Der Prozess umfasste direktes Prompting, Fehleridentifikation und Refaktorisierung.

Der Ausgangspunkt des Projekts war ein einfacher Prompt: „Programmiere in JAVA eine Studierendenverwaltung“. Die initiale Antwort von GPT-4 war die Generierung einer Java-Klasse „Studentenverwaltung“ mit einer privaten Klasse Student, Methoden für das Hinzufügen und Anzeigen von Studierenden (addStudent, showStudents) sowie eine main-Methode mit Scanner, System.out und Schleife für die Ein- und Ausgabe. Diese grundlegende Implementierung stellte einen funktionalen Startpunkt dar, war jedoch noch nicht gemäß dem MVC-Designmuster inkl. Services strukturiert. Daraufhin erfolgte eine Serie von Prompts, um das MVC-Designmuster zu implementieren. Die Anweisungen beinhalteten die Aufforderung zur Trennung von Modell, Controller und View in separate Klassen sowie später auch für Services für Single Responsibility in der zweiten Refaktorisierungsphase. GPT-4 reagierte auf diese Herausforderung, indem es eine differenziertere Struktur mit separaten Klassen für jede dieser Komponenten generierte. So entstanden Klassen für ein Studenten-Modell, einen Controller zur Studierendenverwaltung (mit Methoden wie addStudierende und getStudierendeList) sowie eine Main-Klasse als View. So wurde bei der Kompilierung ein Fehler aufgrund eines fehlenden Imports identifiziert. Die Korrektur erfolgte durch unverändertes Prompting des Terminal-Fehlers, woraufhin GPT-4 den Fehler korrigierte.

Die zweite Refaktorisierungsphase konzentrierte sich auf die Anwendung von Software-Engineering-Prinzipien wie Separation of Concerns und Single Responsibility. Auf weitere Prompts hin entwickelte GPT-4 erweiterte Klassenstrukturen, die diese Prinzipien reflektierten. Es entstanden separate Klassen für Modell, Service, Controller, View (mit Methoden wie printStudierendeDetails), UserInputHandler sowie die Main-Klasse.

Die Ergebnisse zeigen, dass GPT-4 effektiv in der Lage ist, auf direkte Prompts zu reagieren und strukturierten, funktionalen Code zu generieren. Die Notwendigkeit präziser, kontextspezifischer Prompts wurde jedoch auch deutlich, da die Qualität des generierten Codes direkt von der Klarheit und Spezifität der Anweisungen abhängt. Die Fähigkeit des Modells, auf Feedback und spezifische Fehlerhinweise einzugehen, ist ein wertvolles Merkmal, das den Entwicklungsprozess beschleunigen kann. Allerdings unterstreichen die Ergebnisse auch die Notwendigkeit menschlicher Expertise im Prozess, insbesondere in den späteren Phasen der Qualitätssicherung und Feinabstimmung. GPT-4 kann als Hilfsmittel eingesetzt werden kann, um Entwickler:innen repetitive und strukturgebende Aufgaben abzunehmen, während kritische Entscheidungen und komplexere Designaufgaben weiterhin menschlicher Expertise bedürfen.

3.3 Case 2: Webbasiertes Spiel „Snake“

Die Entwicklung des webbasierten Spiels „Snake“ stellt eine umfangreichere Case Study dar, die sich über vier Konversationen mit GPT-4 erstreckte. Eine zentrale

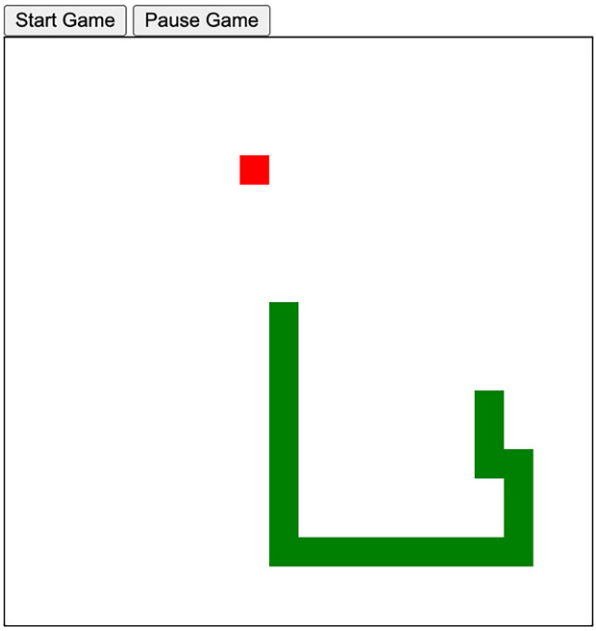

Abb. 1 Screenshot des erstellten Spiels „Snake“ mit Web-Technologien durch GPT-4

Herausforderung dabei war, dass GPT-4 nicht unmittelbar vollständigen Quellcode lieferte, sondern stattdessen zunächst allgemeine Ansätze und Pseudocode als Antwort auf die Prompts generierte. In Abb. 1 wird ein Screenshot des erstellten Spiels gezeigt.

In der ersten Phase der Entwicklung gab GPT-4 auf einen initialen Prompt hin lediglich einen groben Überblick über die Implementierung. Als Folge darauf zielte ein zweiter Prompt auf die Erstellung einer vollständigen HTML-Datei ab, doch das Modell lieferte nur eine Grundstruktur zurück, ergänzt durch JavaScript-Funktionen, die vorerst nur kommentiert waren. Erst auf wiederholte Anweisungen hin produzierte GPT-4 tatsächlichen Code für die JavaScript-Funktionen, wobei gleichzeitig neue Funktionen definiert wurden, die in den folgenden Prompts weiter ausgearbeitet werden mussten.

Diese iterative Entwicklung führte zu einem zunächst nicht funktionsfähigen Code, was den Beginn einer zweiten Konversationsphase markierte. Hierbei wurde GPT-4 mit dem zusammengesetzten HTML/CSS/JS-Code konfrontiert und mit der Anweisung „the script doesn’t work. analyse“ zur Fehleranalyse aufgefordert. *Das Modell identifizierte daraufhin mehrere Problembereiche*, darunter inkonsistent verwendete Positionierungslogiken in CSS, die Handhabung von Spielfeldgrößen und die Positionierung für Schlange und Essen, das Koordinatensystem, Kollisionserkennung, User Experience und Interaktionsaspekte sowie Hinweise zur Performance und Code-Modularität.

Nach der Korrektur dieser Aspekte war das Spiel erstmals ausführbar, allerdings mit einigen Funktionsfehlern, wie einer diagonal statt vertikal bewegenden Schlange. Weitere Anpassungen führten zur vollständigen Funktionsfähigkeit des Spiels, einschließlich einer Erweiterung um einen Pause-Button neben dem Start-Button.

In der dritten Phase wurde ein Refactoring des Codes unternommen, um den prozeduralen Ansatz in einen objektorientierten zu überführen und dabei Best Practices des Software Engineerings wie das SOLID-Prinzip zu berücksichtigen. Dies

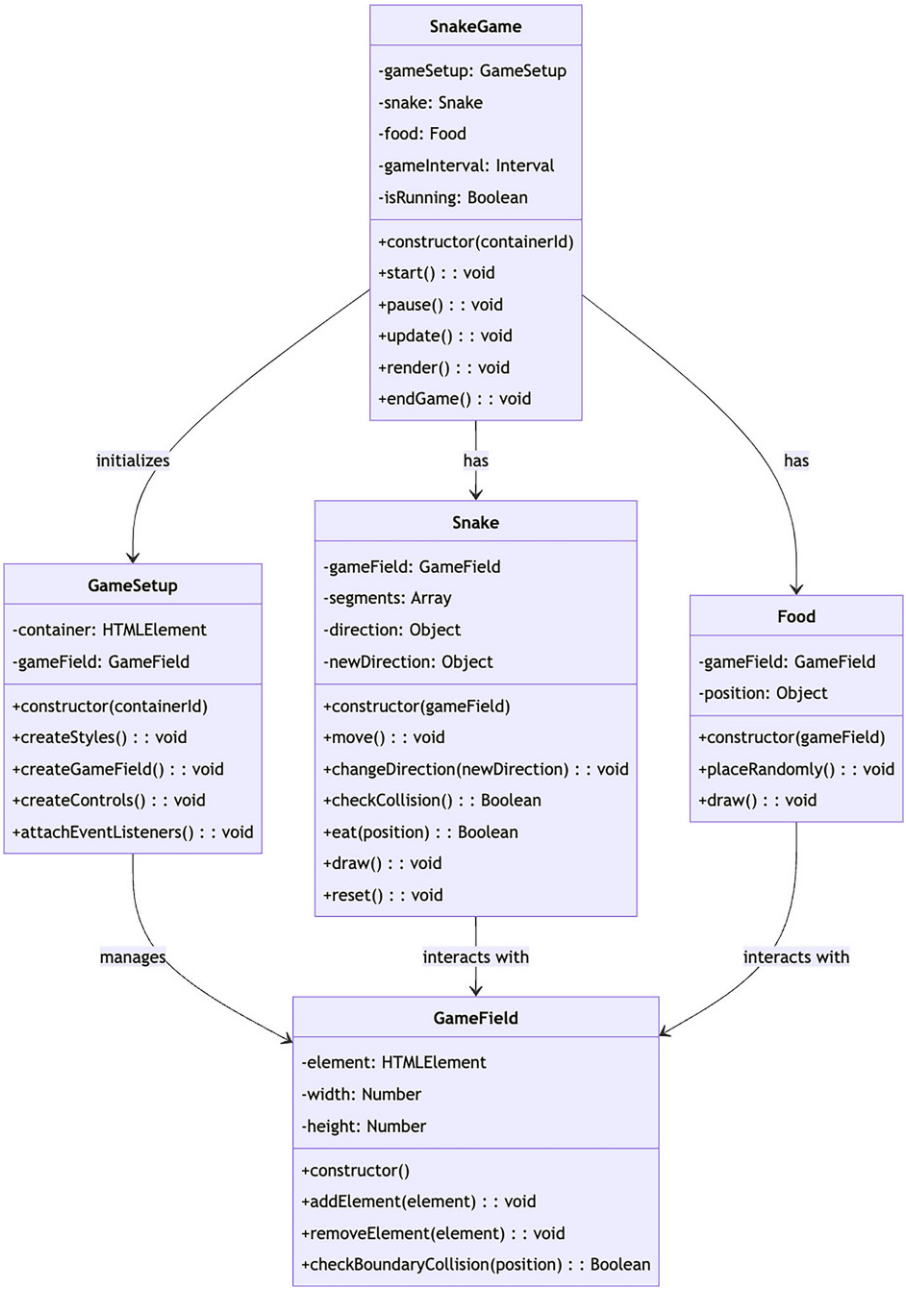

Abb. 2 Refaktorisiertes UML-Klassendiagramm für Snake von GPT-4 erzeugt

beinhaltete auch die Übersetzung von Denglish in standardisiertes Englisch und die Entkopplung von HTML/CSS und JavaScript-Code. Klassen wie Game, Snake und Food wurden identifiziert und implementiert, wobei eine Endlosschleife von GPT-4 korrigiert werden musste, um das Spiel wieder funktionsfähig zu machen.

In der vierten und letzten Phase lag der Schwerpunkt auf objektorientierter Modellierung, wobei UML-Klassen-, Zustands- und Sequenzdiagramme erstellt und mit Mermaid.js visualisiert wurden. Trotz anfänglicher Fehler in den Sequenzdiagrammen, vor allem bei der Kollisionserkennung und Interaktionen mit den Endbenutzer:innen, wurden diese korrigiert. GPT-4 lieferte zudem ausführliche Erläuterungen

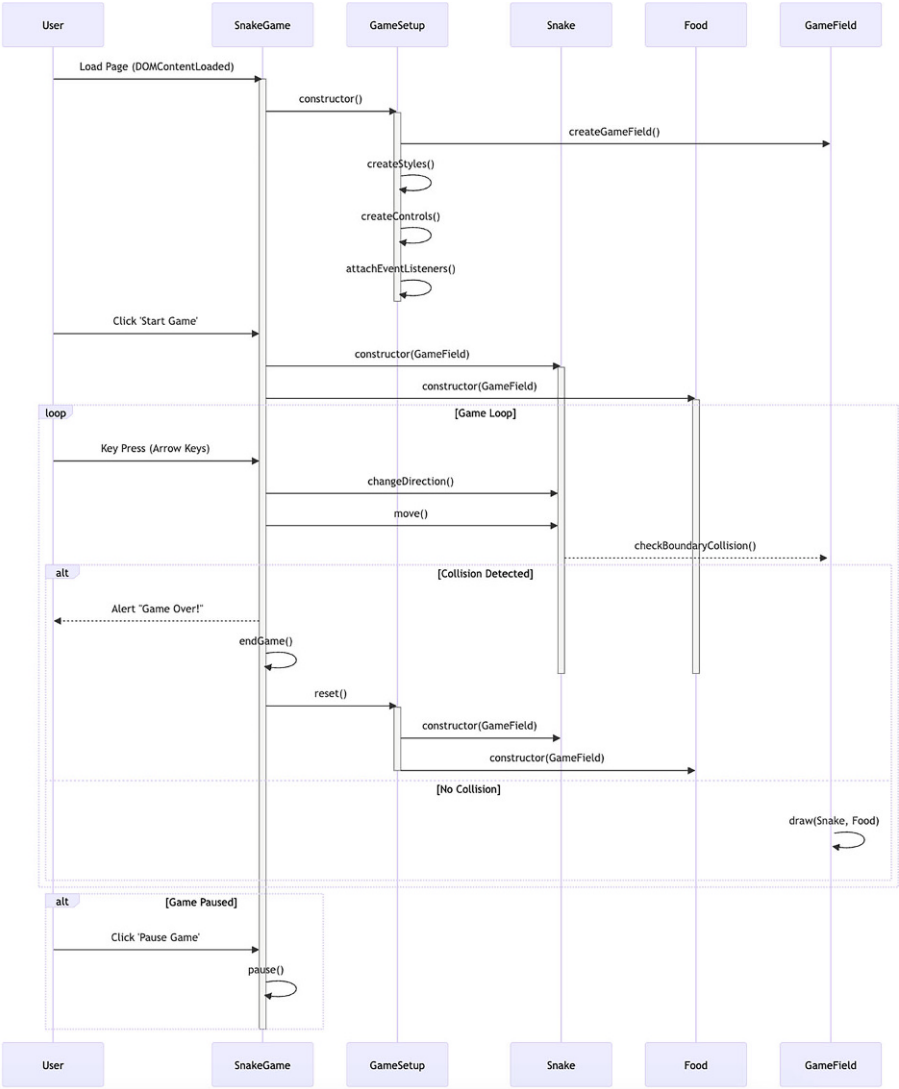

Abb. 3 Refaktorisiertes UML-Sequenzdiagramm für Snake von GPT-4 erzeugt

zu den Prinzipien des Single Responsibility Principle, Open-Closed Principle, Kapselung und Modularität, was zu einem verbesserten Klassendiagramm führte (siehe Abb. 2). Der abschließende Prompt forderte eine Aktualisierung des Zustands- und Sequenzdiagramms (siehe Abb. 3), was die Entwicklung des Snake-Spiels abschloss.

4 Ergebnisse

Die Analyse der Softwarequalität im Kontext der Nutzung von GPT-4 in der objektorientierten Programmierung offenbart sowohl Potenziale als auch Grenzen dieses fortschrittlichen Sprachmodells. Die Studie zur textbasierten Studierendenverwaltung in Java zeigt eindrücklich, dass GPT-4 durch gezielte Anweisungen und präzises Prompting in der Lage ist, funktionale und strukturierte Code-Lösungen zu generieren. Besonders bemerkenswert ist die Fähigkeit des Modells, etablierte Design-Patterns wie MVC zu integrieren und effizient auf Fehlerhinweise zu reagieren, was seine Eignung für die frühen Phasen der Softwareentwicklung unterstreicht. Allerdings wird gleichzeitig die Notwendigkeit deutlich, die Prompts präzise und kontextreich zu gestalten, um qualitativ hochwertige und wartbare Softwarelösungen zu entwickeln.

Diese Ergebnisse implizieren, dass GPT-4 ein effektives Werkzeug zur Unterstützung des Softwareentwicklungsprozesses sein kann, wenn es mit einer durchdachten Prompting-Strategie und fundiertem Verständnis der Softwareentwicklungsprinzipien eingesetzt wird. Jedoch hebt die Studie auch hervor, dass die Anwendung von GPT-4 in der Softwareentwicklung spezifische Herangehensweisen erfordert, insbesondere wenn es um komplexere Programmieraufgaben geht. Die Begrenzungen von One- oder Few-Shot-Prompts bei der Erzeugung funktionsfähigen oder sauberen Codes zeigen, dass das erweiterte Kontextfenster allein nicht ausreicht, um anspruchsvolle Softwareprojekte zu bewältigen. Das webbasierte Spiel hatte trotz des erweiterten Kontextfensters von 128k Token keinen qualitativen Vorteil gegenüber des 16k Kontextfensters, sondern musste durch die Fehleranalyse korrigiert werden.

Die Untersuchungen von Abukhalaf et al. (2023), die sich mit der Anwendung von Prompt Engineering Techniken bei Object-Constraint Languages befassen, bestätigen ähnliche Herausforderungen und unterstreichen die Bedeutung von gezieltem Prompting und der Integration von Software-Engineering-Prinzipien. Zudem wird betont, dass GPT-4 in seiner aktuellen Form automatisierte Tests nicht selbstständig ohne Prompting implementiert, was eine bedeutende Einschränkung darstellt. Diese Erkenntnis hebt die Notwendigkeit menschlicher Intervention und Überwachung in der Softwareentwicklung hervor, insbesondere bei der Planung, der Qualitätskontrolle und der Durchführung komplexerer Designaufgaben.

Abschließend deutet die vierte Konversation der Studie darauf hin, dass GPT-4 ungenutztes Potenzial in der Modellierung besitzt, wenn spezifische Prompts verwendet werden. Dies untermauert die Notwendigkeit weiterführender Forschung, um das volle Potenzial von GPT-4 in der Softwareentwicklung auszuschöpfen und gleichzeitig die zentrale Rolle von IT-Professionals im Designprozess zu bewahren.

5 Diskussion

5.1 Vorgehensmodelle im Software-Engineering und Multi-Agenten-Systeme

Die Studie verdeutlicht, dass die Entwicklung funktionsfähiger Softwaresysteme mit GPT-4 mittels eines Trial-and-Error-Ansatzes möglich ist. GPT-4 besitzt das Potenzial, qualitativ hochwertigen Quellcode zu generieren, allerdings erfordert dies eine gezielte Aktivierung des im Sprachmodell vorhandenen Wissens. Hierzu ist der Rückgriff auf etablierte Methoden und Techniken des Software-Engineerings, insbesondere auf strukturierte Vorgehensmodelle, empfehlenswert. Hong et al. (2023) implementierten ein solches Modell, indem sie GPT-4 in ein Multi-Agenten-System integrierten, anstatt es als Prompt Design Pattern Katalog, wie von White et al. (2023) in Anlehnung an Gamma et al. (1994) vorgeschlagen, zu verwenden. Hong et al. (2023) implementieren für jede Phase des Wasserfallmodells einen Agenten mit umfangreichem Prompt als Zielbeschreibung. Ähnlich agieren Josifoski et al. (2023) die „Flows“ als Softwarekomponente vorstellen, die Tools über APIs mit konversationsbasierten Entitäten wie ChatGPT verbinden, um so intelligente Systeme zu schaffen. Für (Multi-)Agenten, die jeweils mittels Sprachmodelle komplexe Softwaresysteme kollaborativ planen, designen, programmieren und testen, konzipieren Shapiro et al. (2023) eine geschichtete Referenzarchitektur für die Agenten als „autonome kognitive Entitäten“.

Diese aktuellen Forschungen, die sich ebenso wie Techniken und Methoden des Prompt Engineerings, größtenteils in Preprints und Developer-Blogposts finden, deuten darauf hin, dass für die Entwicklung komplexerer Softwaresysteme einfache Konversationen, wie sie in dieser experimentellen Studie verwendet wurden, nicht mehr ausreichend sind. Zudem ist die Komplexität der untersuchten Cases gering im Vergleich zu typischen Entwicklungsaufgaben in der Industrie, was die Aussagekraft der Studie einschränkt und weiteren Forschungsbedarf aufzeigt. Daher wird vorgeschlagen, Vorgehensmodelle des Software-Engineerings als Multi-Agenten-Systeme umzusetzen. Lu et al. (2023) skizzieren hierfür eine weiterführende, konzeptuelle Referenzarchitektur, die über die Ansätze von Hong et al. (2023) hinausgeht. Diese Entwicklungen unterstreichen die Notwendigkeit, Techniken und Methoden des Software-Engineerings zu adaptieren, um das volle Potenzial von LLMs auszuschöpfen.

5.2 Bias in der Anwendung von GPT-4

Bei der Analyse der Reaktionen von GPT-4 auf verschiedene Prompts kann eine Tendenz zur Verzerrung auftreten. Ein bezeichnendes Beispiel hierfür ist die unkommentierte Umwandlung der genderneutralen Formulierung „Studierenden“ im ersten Prompt in „Studenten“. Dieser scheinbar kleine Unterschied offenbart eine tiefgreifende Problematik: die Neigung des Modells voreingenommene Muster zu reproduzieren.

Diese Problematik wird auch von OpenAI selbst erkannt und dokumentiert (OpenAI 2023b). In ihrer öffentlichen Dokumentation, jedoch nur bei Embeddings, stellen sie fest:

„we found that our models more strongly associate (a) European American names with positive sentiment, when compared to African American names, and (b) negative stereotypes with black women."

Dieses Zitat unterstreicht die Bedeutung der sorgfältigen Überwachung und Anpassung der von KI-Systemen generierten Inhalte, um unerwünschte Verzerrungen und Stereotypisierungen zu vermeiden.

Der Bias in KI-Systemen ist nicht nur eine Frage der Genauigkeit oder der technischen Effizienz, sondern auch eine tiefgreifende ethische Herausforderung. Er spiegelt bestehende gesellschaftliche Ungleichheiten wider und kann diese weiter verstärken, wenn er nicht aktiv adressiert wird. Daher ist es von entscheidender Bedeutung, dass Entwicklende und Nutzende von KI-Systemen sich der potenziellen Bias-Problematik bewusst sind und Strategien entwickeln, um diese zu minimieren. Die Integration von KI-Ethik in die Ausbildung und Praxis von Softwareentwicklenden ist dabei ein unverzichtbarer Schritt. Nur durch ein umfassendes Verständnis der Ursachen und Auswirkungen von Bias können wir sicherstellen, dass die Technologien der Künstlichen Intelligenz die Vielfalt und Gerechtigkeit in unserer Gesellschaft fördern, anstatt sie zu untergraben.

6 Fazit

Die vorliegende Studie bietet einen Einblick in die Anwendung von GPT-4 in der objektorientierten Softwareentwicklung und beleuchtet sowohl dessen Potenziale als auch Grenzen. Durch experimentelle Case Studies wurde deutlich, dass GPT-4 mit sorgfältigem und präzisem Prompting effektiv strukturierten und funktionalen Code generieren kann. Dies unterstreicht die Relevanz von LLMs als Hilfsmittel in der Softwareentwicklung, insbesondere in den frühen Phasen der Code-Generierung und beim Prototyping.

Gleichzeitig heben die Ergebnisse die Notwendigkeit menschlicher Expertise im gesamten Entwicklungsprozess hervor. Kritische Entscheidungen, feinere Designaufgaben und die Qualitätskontrolle erfordern weiterhin die Einbeziehung menschlicher Entwickler:innen, um qualitativ hochwertige und wartbare Softwarelösungen zu erstellen. Die Limitationen von GPT-4 in der Bewältigung komplexer Softwareprojekte ohne detaillierte Anleitung verdeutlichen, dass das Modell derzeit eher als Unterstützungswerkzeug denn als alleinstehende:r Entwickler:in fungiert. Erste Lösungsansätze mit Multi-Agenten erzielen bessere Ergebnisse in der Umsetzung komplexer Softwaresysteme.

Die Untersuchung der ethischen Aspekte, insbesondere der Verzerrungen und Stereotypisierungen in den Antworten von GPT-4, wirft wichtige Fragen bezüglich der Nutzung von KI in der Softwareentwicklung auf. Dies betont die Notwendigkeit eines kritischen Bewusstseins für potenzielle Bias und die Entwicklung von Strategien zu deren Minimierung. Die Integration von KI-Ethik in die Ausbildung und Praxis ist dabei unerlässlich, um zu gewährleisten, dass KI-Technologien die Vielfalt und Gerechtigkeit in unserer Gesellschaft fördern.

Weiterführender Forschungsbedarf besteht im Bereich des Prompt Engineering und des Einsatzes autonomer Multi-Agenten in der Softwareentwicklung durch die Integration von KI. Ein systematischer Literaturüberblick über Techniken und Methoden im Bereich des Prompt Engineering könnte tiefergehende Einblicke in effektive Ansätze und Best Practices liefern. Ebenso ist die Erforschung des Potenzials von KI-gesteuerten Multi-Agenten-Systemen zur Softwareentwicklung ein vielversprechender Ansatz, der das Verständnis der Interaktion zwischen KI und menschlichen Entwickler:innen weiter vertiefen könnte.

Funding Open Access funding enabled and organized by Projekt DEAL.

Literatur

Abukhalaf S, Hamdaqa M, Khomh F (2023) On codex prompt engineering for OCL generation: an empirical study. In: IEEE/ACM 20th International Conference on Mining Software Repositories (MSR) https://doi.org/10.1109/MSR59073.2023.00033

Gamma E, Helm R, Johnson R, Vlissides J, Patterns D (1994) Elements of reusable object-oriented software. Design patterns

Hong S, Zheng X, Chen J et al (2023) Metagpt: meta programming for multi-agent collaborative framework. arXiv preprint arXiv:2308.00352

Josifoski M, Klein L, Peyrard ML et al (2023) Flows: building blocks of reasoning and collaborating AI. arXiv preprint arXiv:2308.01285

Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y (2022) Large language models are zero-shot reasoners. Adv Neural Inf Process Syst 35:22199–22213

Lu Q, Zhu L, Xu X, Xing Z, Harrer S, Whittle J (2023) Building the future of responsible AI: a pattern-oriented reference architecture for designing large language model based agents. arXiv preprint arXiv:2311.13148

Mendes A (2023) Chat GPT-4 turbo prompt engineering guide for developers. https://www.imaginarycloud.com/blog/chatgpt-prompt-engineering/. Zugegriffen: 21. Juni 2023

OpenAI (2023a) Best practices for prompt engineering with OpenAI API—How to give clear and effective instructions to GPT-3 and Codex. https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api. Zugegriffen: 21. Juni 2023

OpenAI (2023b) Documentation—capabilities—embeddings. https://platform.openai.com/docs/guides/embeddings/. Zugegriffen: 1. Nov. 2023

Radford A, Narasimhan K, Salimans T, Sutskever I (2018) Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf. Zugegriffen: 1. Nov. 2023

Selvaraj J (2023) Acing the best practices for prompt engineering ChatGPT developers. https://www.c-sharpcorner.com/article/acing-the-best-practices-for-prompt-engineering-chatgpt-developers/. Zugegriffen: 21. Juni 2023

Shapiro D, Li W, Delaflor M, Toxtli C (2023) Conceptual framework for autonomous cognitive entities. arXiv preprint arXiv:2310.06775

Swaylu R (2023) 8 best practices for prompt engineering. https://promptsninja.com/featured/8-best-practices-for-chatgpt-rompt-engineering/. Zugegriffen: 21. Juni 2023
Teubner T, Flath CM, Weinhardt C, van der Aalst W, Hinz O (2023) Welcome to the Era of ChatGPT et al. Bus Inf Syst Eng 65:95–101. https://doi.org/10.1007/s12599-023-00795-x
Vaswani A, Shazeer N, Parmar N et al (2017) Attention is all you need. Adv Neural Inf Process Syst 30:
Wei J, Wang X, Schuurmans D et al (2022) Chain-of-thought prompting elicits reasoning in large language models. Adv Neural Inf Process Syst 35:24824–24837
White J, Fu Q, Hays S et al (2023) A prompt pattern catalog to enhance prompt. engineering with chatgpt. arXiv preprint arXiv:2302.11382
Yang S (2023) Best practices in prompt engineering. https://towardsdatascience.com/best-practices-in-prompt-engineering-a18d6bab904b. Zugegriffen: 21. Juni 2023
Zhang Z, Zhang A, Li M, Smola A (2022) Automatic chain of thought prompting in large language models. arXiv preprint arXiv:2210.03493