Was heisst "Programmieren" im Zeitalter von LLM-basierten Programmier-Assistenten?

Positionsbeitrag

Thomas R. Gross[1]

Abstract: Auf "Large Language Models" (LLMs) basierende Programmier-Assistenten, wie z.B. GitHub Copilot, AWS Code Whisperer, oder Google Bard stellen die (universitäre) Lehre der Informatik vor neue Herausforderungen. Wenn wir jetzt das "End of Programming" erreicht haben, warum sollen dann Studierende noch Programmieren lernen (und Dozierende dieses Fach unterrichten)? Während verschiedene Beiträge die Reaktionen der Informatik Dozierenden dokumentieren (und die Auswirkungen solcher Werkzeuge auf formative und summative Evaluation untersuchen), so ist auch eine Diskussion über die inhaltlichen Folgen dieser Werkzeuge nötig. In diesem Beitrag argumentiere ich, dass bei Einsatz dieser Werkzeuge (i) Programmieren weiterhin notwendig für die Ausbildung in Software Engineering (und anderen Gebieten der Informatik) ist, und (ii) Programmieren als Grundlage für die Entwicklung von Software Systemen weiterhin relevant ist und als solche unterrichtet werden sollte.

Keywords: LLM-basierte Werkzeuge; Programmieren; Software Technology

1 Einleitung

In den letzten 2 - 3 Jahren sind Programmier-Assistenten, die auf Large Language Models (LLMs) basieren, so weit fortgeschritten, dass sie in gängige Entwicklungsumgebungen (IDEs) integriert werden konnten und damit das Codieren in vielfacher Hinsicht erleichtern. Basierend auf einer grossen Anzahl früher geschriebener Programme können diese Werkzeuge in vielen Situationen Vorschläge machen, und die Benutzer/innen solcher Werkzeuge geben dann Hinweise (oder weitere Anforderungen) ein, um das erwünschte Programm zu erstellen. Beispiele solcher LLM-basierten Werkzeuge sind GitHub Copilot, AWS Code Whisperer, Google Bard oder Meta Code Llama.

Im folgenden beziehe ich mir (nur) auf GitHub Copilot, dass nach den Beobachtungen anderer Dozierender durch verschiedene Ansammlungen von Informatik Programmierproblemen trainiert wurde[Be22]. Insbesondere passen "klassische" Programmierprobleme (die in vielen Büchern zur Einführung in die Programmierung verwendet werden) gut zum Lösungsansatz solcher Werkzeuge, und verschiedene Projekte haben die Fähigkeiten dieser Werkzeuge dokumentiert [DKG23, We23b, BJP23]. Prüfungen in einem solchen Umfeld

[1] ETH Zürich, Departement Informatik, 8092 Zürich, Schweiz

doi:10.18420/seuh2024_07

sind daher eine Herausforderung für universitäre Programmierkurse[CA23, Ou23] und die Dozierenden verschiedener Institutionen haben unterschiedlichste Strategien zur Integration (oder dem Verbot) dieser Werkzeuge vorgeschlagen[LG23].

Allerdings wird durch die Programmier-Assistenz eine Lösung oft nicht sofort (nach Eingabe der Anforderungen, d.h. der zu lösenden Aufgabe) erstellt, sondern die Benutzer/innen müssen die Programmier-Assistenz anleiten, eine vollständige richtige Lösung zu erstellen[2]. Im Raum steht die Behauptung, dass diese Werkzeuge das "End of Programming" zur Folge haben [We23a] und Programmierkenntnisse für zukünftige (und jetztige) Informatiker nicht mehr relevant sind. Niemand wird Software warten müssen – wenn eine neue Herausforderung identifiziert ist (oder ein neuer Bug entdeckt wurde), dann wird das Programm neu mit einem Werkzeug generiert (und nicht durch Modifikation des bestehenden Programms realisiert).

2 Das "End of Programming"?

LLM-basierte Werkzeuge nutzen aus, dass es für ein gegebenes Problem (oder zumindest für ein sehr ähnliches Problem) bereits (Teil)Lösungen gibt, die dann zu einer Lösung für das gegebene Problem kombiniert werden können. Für viele der Probleme, die in Programmierkursen verwendet werden, sind natürlich die Lösungen bekannt, und so haben diese Werkzeuge keine grosse Mühe, eine Lösung zu finden. Aber für viele reale Projekte ist eine grosse Code Basis nicht vorhanden, so dass es viel zu früh ist, vom "End of Programming" zu sprechen[Ye23].

Auch unsere Erfahrung mit GitHub Copilot unterstützt die Einschätzung, dass der Nachruf auf Programmieren verfrüht ist. Tabelle 1 zeigt wieweit Copilot die wöchentlichen Prüfungsaufgaben eines Semesters einer "Einführung in die Programmierung" Vorlesung (die Java als Programmiersprache verwendet) für das 1. Semester lösen konnte. Die 1. Spalte gibt an, in welcher Semesterwoche ein Problem gestellt wurde. Die 2. Spalte gibt das Thema der Aufgabe an. Keine dieser Aufgaben konnte Copilot direkt *ohne weiteren Input* (d.h. ohne Hinweisungen und Anweisungen) lösen. Spalte 3 zeigt an, welchen Prozentsatz der Unit Tests das mit Hilfe von Copilot innerhalb nützlicher Zeit erstellte Programm korrekt ausführen konnte[3].

Es ist nicht überraschend, dass Copilot mit Hinweisen und Anweisungen in der Lage ist, die Aufgaben zu lösen. Aber die Hinweise und Anweisungen setzen profunde Programmierkenntnisse voraus. Insbesondere mussten die Benutzer/innen Hinweise zur Wahl

[2] Diese Einschätzung wird auch von den Entwicklern Werkzeuge geteilt; GitHub Copilot wird als "Your AI *Pair Programmer" beworben.*

[3] Die Programmieraufgaben wurden auf Deutsch gestellt. Frühere Experimente (mit den Sprachen Deutsch und Italienisch) bestätigten, dass die Wahl einer (gängigen) Sprache für die Aufgabenstellung nicht relevant ist. Die automatische Spracherkennung und anschliessende Übersetzung sind gut genug für die LLM-basierte Programmierassistenz.

Woche	Prozentsatz Unit Tests	
W4	Arrays, control flow	100
W5	Data structures, control flow	100
W6	Arrays, recursion	66.67
W7	Graph manipulation, references	100
W8	Paths, references, recursion	100
W9	Objects, simulation, inheritance	100
W10	Collection Framework , exceptions	100
W11	Collection Framework, object design	100
W12	Maps, object design, overriding	32.58

Tab. 1: Bearbeitungserfolg von Copilot für verschiedene Prüfungsaufgaben.

der Datenstrukturen (bzw. Darstellung) geben. Diese Anforderung kann natürlich auch eine Nebenwirkung der Aufgabenstellungen sein, die die Verwendung des Java Collection Frameworks betonen.

Wenn also Programmierer/innen mit diesen Werkzeugen erfolgreich arbeiten wollen, dann müssen sie weiterhin Programmierkenntnisse haben – und da Copilot das volle Spektrum der Programmiersprachen (in unserem Fall: Java) nutzt, müssen Programmier/innen mit allen Konzepten (insbesondere des Objektsystems) vertraut sein. Diese Kenntnisse sind auch erforderlich, wenn man den Output von Copilot überprüfen möchte, z.B., um festzustellen, dass das erstellte Programm nicht unerwünschterweise Informationen weitergibt (wobei es keine Rolle spielt, ob solches Verletzen der Erwartungen absichtlich oder unabsichtlich gemacht wird).

3 Was heisst “Programmieren”?

Der Einsatz von Programmier-Assistenten wie Copilot kann Programmieren interessanter machen, da die Assistenz manche wichtige aber letztlich unkritische Aufgabe übernimmt. Wenn man für eine gegebene Klasse den Konstruktor finden muss, der am zweckmässigsten die Objektexemplare initialisiert, dann macht Copilot oft einen guten Vorschlag (und ist weniger mühsam als aus den Optionen der Code Completion allein (oder gar der API) die beste Auswahl zu treffen). Copilot kann also eine grosse Hilfe für den Schritt des Codierens sein. Aber wenn wir unter “Programmieren” (auch) das Zerlegen eines Problems in Teil-Probleme und das Zusammenfügen der (Teil)-Antworten zur Lösung eines Problems verstehen [Ce16] dann hilft Copilot aber löst nicht das Problem des “Programmierens”.

Copilot ist auch nur begrenzt hilfreich im kritischen Lesen der Aufgabenstellungen. Eine Programmier-Assistenz kann helfen, verschiedene Varianten (eines Programms zu erstellen), aber welche der Varianten denn die Anforderungen am besten erfüllt (und welche Aspekte in der Aufgabenstellung noch festgelegt werden müssen, um das Verhalten genügend genau

zu spezifizieren) ist eine Frage, für die ein Programmgenerator nur bedingt geeignet ist. Es wäre daher wünschenswert wenn Veranstaltungen, die das Programmieren lehren, verstärkt diese Aspekte der Software Technologie betonen um Studierende auf eine produktive Arbeit mit Programmier-Assistenzen vorzubereiten.

4 Bemerkungen

LLM-basierte Programmmier-Assistenten bieten eine Möglichkeit, das Programmieren interessanter zu machen. Aber die Interaktion mit solchen Werkzeugen setzt noch immer voraus, dass die Benutzer/innen programmieren können. Programmieren ist daher weiterhin notwendig für die Ausbildung in Informatik und insbesondere in Software Engineering. Ein wichtiger Teil des "Programmierens" ist das Erstellen (und Überprüfen) von Anforderungen. Programmieren ist eine Grundlage für den Entwurf und die Entwicklung von Software Systemen und die Ausbildung sollte diesem Umstand Rechnung tragen.

Literaturverzeichnis

[Be22] Berger, E.: , Coping with Copilot. Blog, Aug 2022.

[BJP23] Barke, Shraddha; James, Michael B.; Polikarpova, Nadia: Grounded Copilot: How Programmers Interact with Code-Generating Models. Proc. ACM Program. Lang., 7(OOPSLA1):85–111, 2023.

[CA23] Cipriano, Bruno Pereira; Alves, Pedro: GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2023, Association for Computing Machinery, New York, NY, USA, S. 61–67, 2023.

[Ce16] Cerf, Vinton G.: Computer Science in the Curriculum. Commun. ACM, 59(3):7, feb 2016.

[DKG23] Denny, Paul; Kumar, Viraj; Giacaman, Nasser: Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, S. 1136–1142, 2023.

[LG23] Lau, Sam; Guo, Philip: From "Ban It Till We Understand It"to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. In: Proc. ICER'23. ACM, August 2023.

[Ou23] Ouh, Eng Lieh; Gan, Benjamin Kok Siew; Jin Shim, Kyong; Wlodkowski, Swavek: ChatGPT, Can You Generate Solutions for My Coding Exercises? An Evaluation on Its Effectiveness in an Undergraduate Java Programming Course. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. ITiCSE 2023, Association for Computing Machinery, New York, NY, USA, S. 54–60, 2023.

[We23a] Welsh, Matt: The End of Programming. Commun. ACM, 66(1):34–35, 2023.

[We23b] Wermelinger, Michel: Using GitHub Copilot to Solve Simple Programming Problems. In (Doyle, Maureen; Stephenson, Ben; Dorn, Brian; Soh, Leen-Kiat; Battestilli, Lina, Hrsg.): Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 1, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023. ACM, S. 172–178, 2023.

[Ye23] Yellin, Daniel M.: The Premature Obituary of Programming. Commun. ACM, 66(2):41–44, 2023.