Integration of Chatbots for Generating Code Into Introductory Programming Courses

Olaf Herden

Cooperative State University Baden-Wuerttemberg, Germany

Abstract

Recently, generative artificial intelligence made a great stir. These systems have the ability to generate text, images or other data using generative models. They will influence all kind of knowledge work, especially the way of writing computer programs. For software developers the way of working will change by using these systems. Therefore, it is necessary to show students the use, possibilities and limitations of these techniques.
In this contribution we report about an experience, where we have used the chatbot ChatGPT 3.5 for lab exercises in the introductory programming course. The chatbot was used e.g. for generating, explaining and simplifying code. Beside predefined exercises the students also defined programming problems on their own. Based on these tasks the students generated prompts and sent them to ChatGPT. The results were critically checked, evaluated and annotated.
In summary, the quality of generated code or explaining code was surprisingly good, especially for small and common problems. It is obviously, that developers will delegate simple and repeating coding tasks to chatbots or other generative artificial intelligence systems in the future. For these reasons it is important to integrate working with generative artificial intelligence systems into introductory programming courses. The advantages and drawbacks of this approach should be imparted to students and they should get an impression about future software development.

Keywords: *higher education, computer science, programming, generative artificial intelligence, chatbot*

1. Introduction

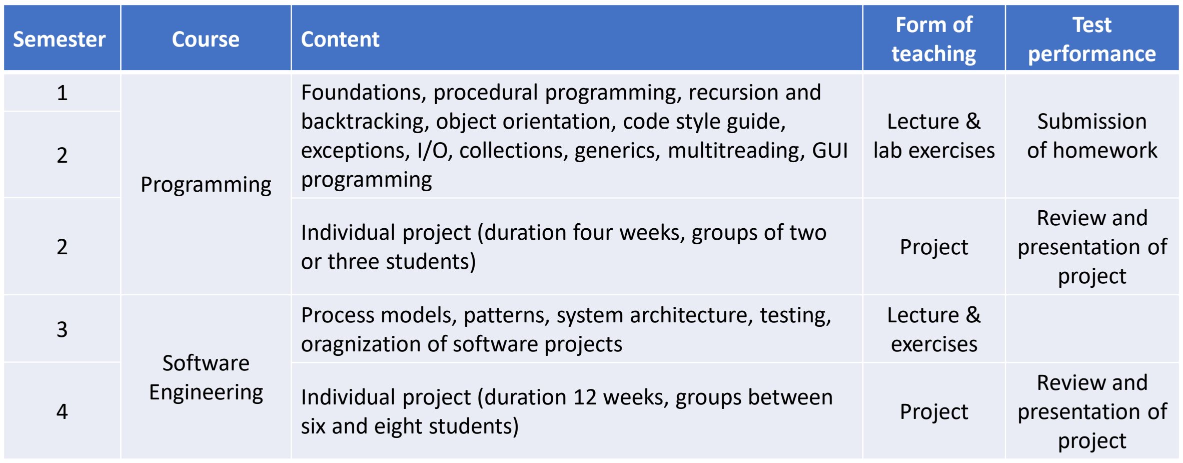
Programming is an important skill of computer scientists. Therefore, learning to develop programs for a given problem and implement them into a programming language is part of every undergraduate curriculum in computer science.
The typical structure of this course is a mixture of classical lectures combined with comprehensive coding exercises in the lab.
In recent years, AI (artificial intelligence) made a tangible progress. This development is based on modern hardware architectures and the improvement of known or the invention of new algorithms. One special kind of AI is GenAI (generative AI) [1]. GenAI systems have the ability to generate text, images or other data using generative models [2]. Typically, users enter a prompt, an input in terms of a question or a request [3] and the system responds the generated answer.
These systems affect all kind of knowledge work and especially influences software development [4, 5]. Therefore, it is necessary to show students the use, possibilities and limitations of these techniques [6].
In this contribution we want to discuss the use of chatbots in introductory programming lectures and we want to answer the questions:

- Can LLM based chatbots be used for solving typical exercises in introductory programming courses?
- Can LLM based chatbots be used for explaining solutions of these exercises?
- Can LLM based chatbots act as a tutor additional to the lecturer?
- How is the degree of automation in programming when using a chatbot?

The remainder of the contribution is organized as follows: first we give an overview of teaching programming and software engineering in undergraduate computer science in section 2 and an introduction to GenAI and large language models in section 3. In section 4 we describe how chatbots can be used in programming. Our approach of integrating GenAI into the programming lecture is sketched in section 5. We evaluate this approach in section 6. The paper concludes with a summary and an outlook.

2. Teaching Programming and Software Engineering

The concept of teaching programming and software engineering in undergraduate computer science course of studies is depicted in figure 1. It follows the principle of distinguishing between programming in the small and programming in the large [7].



Semester	Course	Content	Form of teaching	Test performance
1	Programming	Foundations, procedural programming, recursion and backtracking, object orientation, code style guide, exceptions, I/O, collections, generics, multitreading, GUI programming	Lecture & lab exercises	Submission of homework
2				
2		Individual project (duration four weeks, groups of two or three students)	Project	Review and presentation of project
3	Software Engineering	Process models, patterns, system architecture, testing, oragnization of software projects	Lecture & exercises	
4		Individual project (duration 12 weeks, groups between six and eight students)	Project	Review and presentation of project

Fig. 1: Teaching in programming and software engineering

The two-semester programming course and then subsequently software engineering course are obligatory for every student. The programming course teaches programming in the small and its content are foundations of programming, elements of imperative programming languages, the main concepts of object orientation, coding style guides, and an introduction into GUI (graphical user interface) programming. All these topics are taught by a classical lecture followed by comprehensive lab exercises. At the end as test performance there is a programming project where the students work on a problem for about four weeks.
The software engineering course is split into two periods. In the first period the concepts necessary for programming in the large are taught. Basic concepts are process models, software architecture, modeling languages, testing and organizational issues of software development. In the second period the students work on a larger project over one semester in groups having between six and eight student members.

3. Generative Artificial Intelligence and Large Language Models

GenAI is a kind of AI that can produce different kind of content. While there are AI systems that can generate images (e.g. Midjourney), videos or sound, text generating systems (e.g. OpenAI's ChatGPT or Google's Gemini) are at the centre of the current debate.
These systems have a chatbot as frontend to communicate with the user and they are based on an LLM (large language model). LLMs are generative models, they can produce new content based on the data they are trained with. The most famous LLMs are GPT (developed by OpenAI), Gemini (developed by Google), Llama3 (developed by Meta) and Claude 3 (developed by Anthropic).
The user’s interaction with the chatbot is realized by prompts. A prompt is a question or instruction given to the bot. The practice of designing and formulating input that will produce optimal output is often called prompt engineering.

4. Chatbots in Programming

In this section we describe in which way chatbots based on LLMs can be used in programming tasks. Therefore, we have analyzed our lecture material and books for introductory programming courses [8, 9].
We have identified the following seven tasks where chatbots can probably help in the coding process:

- (T1): Explaining basic knowledge: we ask the chatbot a question to explain something, e.g. "What is the IEEE754 standard?"
- (T2): Constructing code: we give a textual description and want the chatbot to give us the code, e.g. "Can you give me a Java implementation for the 8 queens problem"
- (T3): Describing code: the input is a program or code fragment and the chatbot should explain what it does or calculate the output, e.g. "Given the following Java code. What are the values of the variables i and j at the end?"
- (T4): Refactoring code: existing code should be transformed into a better (in terms of readability or performance) version, e.g. "Can the code in method m be transformed into a more understandable version?"
- (T5): Formatting code: given code should be checked for optical aspects like indents or braces and transformed if necessary, e.g. "Format the code in the following class appropriate to the Java style guide"
- (T6): Checking coding style: given code should be checked for programming aspects like naming conventions for variables and classes, e.g. "Are the variable names in the following Java code appropriate?" or should detect so called code smells [10]
- (T7): Commenting code: given code should be supplemented by comments, e.g. "Replenish the following Java class by comments in Javadoc style"

5. Our Approach

Like in most undergraduate courses, in introductory programming courses we always have the problem to teach sound basic knowledge as well as state-of-the-art in e.g. languages, frameworks and tools [11, 12]. The use of chatbots can be seen as a modern tool for software developers. So, we think it is necessary to use it in the courses and to show the students the possibilities and limits of these systems.
For each task (T1) – (T7) we have defined two exercises for the students. On the one hand, they should use two or three examples from the catalogue of existing exercises. On the other hand, the students should define two or three examples on their own. The task should be given as a prompt to the chatbot and the reply should be documented and checked for plausibility and correctness. If the answer is not sufficient, a second and probably a third prompt should be formulated. These prompts should give hints like "My professor said …" or "A classmate told me …". The chat history should be documented.
In the tasks (T2), (T3) and (T4) we distinguish different levels. In (T2) we distinguish between classical standard exercises and self-defined exercises. The expectation is that the answers of the first group are better than of the second group. For code describing (task (T3)) we differentiate between standard algorithms well-known from literature, small self-written programs, code with small imperfections, correct code but violating good code style, logical incorrect but syntactical correct code and not compilable code. For the refactoring task (T4) we decide between good, correct code, code with small imperfections, correct code but violating good code style and logical incorrect but syntactical correct code.

6. Evaluation and Results

A group of 28 students carried out the exercises described in the former section using the free version ChatGPT 3.5. The results can be summarized as follows:

- (T1): Almost all questions about basic knowledge was answered sufficiently by the chatbot, some needed a second prompt.
- (T2): The chatbot was very good in constructing code, for standard algorithms as well as for self-defined problems appropriate solutions were given. In some cases a second prompt was necessary to remove ambiguity.

- (T3): The chatbots' performance in describing given code was varying. If the problem is well-known and the code is well written (i.e. following common code conventions) the code was described correctly in almost all cases. But when using bad coding style (e.g. so called spaghetti code) the chatbot had serious problems to understand and explain the code. In some cases even a second or third prompt did not help to lead the chatbot to a correct answer.
- (T4): In this category the results also depend from the input. In most cases, high quality code was detected and no refactoring applied. When the code had well-known deficits the chatbot detected them and was able to apply common refactoring rules. But in examples with bad code style or logical errors (and these kinds of code exists in real life), the chatbot had some problems and was not able to refactor the code.
- (T5): In formatting code the chatbot reached very good results. This is not very surprising because formatting source code is a task typical done by tools. But the chatbot can also explain bad formatted code fragments and knows different code styles (e.g. Kernighan/Ritchie style, Allman style, Whitesmiths style) and can transform source code between these variants.
- (T6): Also in checking code style the chatbot showed a good performance. Violations against common naming conventions for constants and variables were detected, improvements suggested and explained. However, some more hidden violations of coding conventions were not detected.
- (T7): Adding comments to given source code was done very well by the chatbot. Especially the comments generated for Javadoc had a high quality.

All in all we can say that the chatbot obtained a good result. Some limitations in describing given code and refactoring were observed.
The initial questions can be answered as follows: In many cases LLM based chatbots can

- be used for solving typical exercises in introductory programming courses
- be used for explaining solutions
- act as a tutor

But in all aspects the critical reflection by students and an advisor is still necessary.
For classifying the grade of automation we use the schema about collaboration perspective from [13]. From our point of view today we are somewhere between the stages model in the loop and human in the loop. Model in the loop means that there is a balanced competence partitioning between the human and the AI. A little bit more wide-reaching is the human in the loop approach where the LLM makes automated judgements, the human verifies them afterwards.

7. Summary and Outlook

In this contribution we have covered the use of LLM based chatbots in introductory programming courses. First we have sketched the fundamentals on GenAI and the organizational structure of lectures in programming and software engineering. Then we have identified typical tasks during programming where LLM based chatbots can support the developer. In our practical study students have used the chatbot ChatGPT 3.5 for the different tasks. In summary, the quality of generated code or explaining code was good. Therefore we conclude that developers can delegate some coding tasks to chatbots and the process of developing software will change. So it is important and necessary to integrate working with GenAI systems into introductory programming courses.
For the near future we see some interesting tasks of working with LLM based chatbots in programming:

- In our evaluation we have used ChatGPT 3.5. It would be interesting to use and compare with other tools like Google Gemini (formerly known as Google Bard) or Microsoft Copilot (formerly known as Bing Chat).
- Another interesting tool is GitHub Copilot. It is specialized for coding and it can be used within an IDE (integrated development environment), the main tool in professional software development.
- Observing the development of LLMs. The hypothesis is that the gap between humans and generated solutions will be closing with every new version. Questions are how fast will this gap closing be and how close will future versions come to the best solution.
- It was a conscious decision to use freestyle prompts in our evaluation. It might be interesting to compare the results with more structured prompts.

REFERENCES

[1] S. Feuerriegel, J. Hartmann, C. Janiesch, P. Zschech. Generative AI. Bus. Inf. Syst. Eng. 66(1): 111-126 (2024).
[2] D. Leslie, F. Rossi: *Generative Artificial Intelligence*. Techbriefs, ACM Technology Policy Council. 2023.
[3] T. Teubner, C. Flath, C. Weinhardt, W. van der Aalst, O. Hinz. *Welcome to the Era of ChatGPT et. al*. Business Information Systems Engineering. 65(2), p. 95-101, 2023.
[4] C. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, K. Narasimhan. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?*. Computing Research Repository (CoRR) abs/2310.06770, 2023.
[5] M. Sako: How Generative AI Fits into Knowledge Work. Commun. ACM 67(4): 20-22 (2024).
[6] H. Gimpel, K. Hall, S. Decker et. al. *Unlocking the Power of Generative AI Models and Systems such as GPT-4 and ChatGPT for Higher Education*. White Paper, March 2023.
[7] F. DeRemer, H. Kron. Programming-in-the-Large Versus Programming-in-the-Small. IEEE Trans. Software Eng. 2(2): 80-86, 1976.
[8] R. Sedgewick, K. Wayne. Introduction to Programming in Java: An Interdisciplinary Approach. Sams Publishing, 2nd edition, 2017.
[9] N. Samoylov. Introduction to Programming. Packt Publishing, 2018.
[10] R. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008.
[11] R. Simha, A.N. Kumar, R.K. Raj. Undergraduate Computer Science Curricula. Commun. ACM 67(2): 29-31 (2024)
[12] M. Johnson. Generative AI and CS Education. Commun. ACM 67(4): 23-24 (2024)
[13] G. Faggioli, L. Dietz, C.L.A. Clarke, G. Demartini, M. Hagen, C. Hauff, N. Kando, E. Kanoulas, M. Potthast, B. Stein, H. Wachsmuth. Who Determines What Is Relevant? Humans or AI? Why Not Both? Commun. ACM 67(4): 31-34 (2024).