# Practices for the Application of Generative AI in Programming Education

Christian Köppe, Hieke Keuning, Ioanna Lykourentzou ,Isaac Alpizar Chacon, Sergey Sosnovsky

Utrecht University

In a recent survey, which was distributed as part of the current fUSO project Generative AI in Programming Education, 70% of the responding Information and Computing Sciences teachers indicated that they expect to use GenAI tools in teaching practices in the future. A short inventory of teachers' recent GenAI practices revealed only a limited application yet, which indicates the need for collecting and sharing knowledge on specific practices of using GenAI in Programming Education.

Part of the current fUSO project is the collection of such practices for integrating GenAI in Programming Education. These practices were collected from academic and non-academic resources. For the academic resources, a rapid systematic review was performed which resulted in six publications which contained descriptions of practices (in varied details). During the project, also a list of 50+ non-academic resources was collected in which (potential) practices were identified. The practices are presented here in an aggregated version, containing the core idea, some implementation details and the source/s of the practice. The practices will be described in more detail (probably using the format of educational design patterns) later. Furthermore, some workshop/training material will be developed to support dissemination.

Please note: Practices are updated and subject to change and improvements [current version: 3-6-2024]

All information on the project and additional resources can be found on the project website: https://www.uu.nl/en/research/generative-ai-for-computing-education

## THE PRACTICES

### Category: Students use AI for Learning Material/Activity Creation

#### Practice: Concept teaching/explanation

Instead of asking a human instructor, students can ask the tool to explain a programming concept. This could include the generation of some example code which implements/applies the concept. Students can make this teaching/explanation more context-specific, e.g. by providing some extra

information in the prompt like the task they are working on or elements of the concept they do not understand yet.

To refine the level of teaching/explanation, this practice could be combined with a persona prompt, for example: "From now on, act as computer science teacher teaching programming in xxx at bachelor level." (White et al., 2023)

Examples: "What's the difference between checked and unchecked exceptions in Java? Give code examples for each."

Source: (Lau & Guo, 2023)

## Practice: Exemplar solution generation

Exemplars and worked solutions are a proven way to support student understanding of various concepts, students use them as models when learning. Students can generate such solutions, even including worked examples with additional reasoning for coding decisions etc. AI can also generate a variety of solutions, exposing students to the efficiencies of and differences in writing code in various ways.

Source: (Becker et al., 2023)

## Practice: Explanation of how code works

If students do not understand how a piece of code works, students can ask AI to explain it. This can be done in a simple way, e.g. "Explain what this piece of code does: [student's code]". To get explanations on a certain expertise level, the 'persona prompt pattern' (White et al., 2023) could be included. For instance, "ChatGPT, I want you to take on the persona of a university CS1 instructor talking to a student who has never taken a programming class before. Explain what this function does: [user's code]." Another option is to specify/prime the type of desired code description: high-level, problem statement-like, or step-by-step.

Users can also ask AI to automatically generate code comments or API documentation.

Sources: (Becker et al., 2023; Lau & Guo, 2023; Sarsa et al., 2022)

## Practice: GenAI review of student-written code

AI tools can also serve as a reviewer for code students have written. The tool can be asked to provide general or detailed critiques. Variations would be to ask for an (enumerated list) of critique points so that the student can ask follow-up questions on specific elements of the review/critique.

The 'persona prompt pattern' can be useful here, e.g.,: "I want you to take on the persona of a senior software engineer at a top technology company. I have submitted this code to you for a formal code review. Please critique it: [user's code]"

Sources: (Becker et al., 2023; Lau & Guo, 2023)

## Practice: Creation of extra exercises for students who want more practice.

GenAI can produce novel learning resources from a single priming example including programming exercises. The priming example should include keywords (such as "cars, function, parameters, conditional"), the original problem statement ("write a function called speeding_check that takes a single parameter speed and prints out...), and optionally also a sample solution and test cases. For

the generation of the new exercises, keywords could be used to produce contextualized problem statements targeting certain thematic topics, e.g. by giving keyword “ice hockey” instead of cars. These generated exercises can also include an appropriate sample solution and test cases, similar to the original exercise.

Sources: (Becker et al., 2023; Lau & Guo, 2023; Sarsa et al., 2022)

### Practice: Think-Pair-Share (with AI)

This is a variation of the think-pair-share activity. Students try to solve a programming task (e.g., reversing a string) and then prompt an AI to solve it. Then each pair would discuss how their human solutions compare to the AI solution.

Source: (Lau & Guo, 2023)

### Practice: Review of AI-generated code

Students could turn into code reviewers by taking generated code and analyzing it. They could be asked to highlight errors and inconsistencies and judge the quality of the code according to several criteria. This evaluation of code quality can be used to engage students at the higher levels of Bloom’s taxonomy. This may prove useful for generating discussions around alternative approaches and the quality of solutions and provide the basis for refactoring exercises.

Sources: (Kendon et al., 2023; Lau & Guo, 2023)

## Category: Students use AI for solving errors

During programming, students often have errors in their programs. These can be both syntactical, semantical or both. The following practices support students with solving these errors.

### Practice: Error messages explanation

Programming (compiler) error messages are a known barrier for student progress. Students can ask AI to explain an error message in general. They also can provide the error message and the code and can get a concrete explanation plus correct fixes.

Source: (Becker et al., 2023)

### Practice: Debugging help

Students can ask GenAI to find possible bugs in the given code and explain why those may be bugs. Standard bugs in student code (e.g., offby-one errors in a loop bound) can be found by the tool matching against patterns learned from billions of lines of open-source code. Students can ask follow-up clarifying questions and engage in a back-and-forth debugging conversation.

Example: “Here is my code and the output I see when I run it. This output looks wrong because the last two array elements are duplicated. What should I change in my code to help me find the bug more easily?" Notably, students have to run the code by themselves, potentially leading to a next round of dialogue if the error hasn't been solved sufficiently.

Source: (Lau & Guo, 2023)

## Category: Students use AI for co-creation

### Practice: Co-code (or co-create) a program

Students could have sections of code written by AI and include these into their program to complete a given task. This should include explanations of what amendments they had to make to generate a working version.

Students could do the high-level design and let AI fill in method-level code. Or the students design the algorithm, and the AI fills in the subgoals/steps.

Source: (Kendon et al., 2023)

### Practice: Specification-to-code conversation

Students are often not good at writing precise specifications, so the generated code may not be what is needed. The 'flipped interaction prompt pattern' helps (White et al., 2023) by having a back-and-forth conversation with ChatGPT before it generates the requested code.

Example: "ChatGPT, I want you to write a Python function to join first and last names. Ask me clarifying questions one at a time until you have enough information to write this code for me."

Source: (Lau & Guo, 2023)

### Practice: Code refactoring

Students can use GenAI to help them with refactoring their code. This could be done to improve readability, style, or maintainability. The refactoring results can be improved by holding a conversation with ChatGPT and answering its follow-up questions.

Example: "Refactor this function to use smaller helper functions." or "Rewrite this code using only simple Python features that a student in an introductory programming course would know about."

Notably, students could use this technique to generate more 'plausible-looking' answers to programming assignments, because otherwise it may look suspicious if they turn in code that uses too many advanced language features.

Source: (Lau & Guo, 2023)

### Practice: Exercise quick start (Alleviating programmer's writer's block).

Students sometimes don't know how to get started writing a programming exercise, or to continue once stopped. GenAI can be used to produce starter code, thereby offering the opportunity to extend code rather than struggling with a blank page. This might require to shift the focus towards rewriting, refactoring, and debugging code instead of writing every line of code from scratch.

Source: (Becker et al., 2023)

## Category: Assessment support

These practices can be part of formative and summative assessments.

### Practice: Assessment of student's GenAI prompt quality

The quality of the prompt largely also determines the quality of the generated result. Students can be assessed based on their ability to produce effective AI prompts that generate the best possible solution to a problem. This requires a more thorough problem analysis and consequently a better understanding of the problem-solution space.

Sources: (Kendon et al., 2023; Denny et al., 2024)

### Practice: Assessment of student's analysis of GenAI-generated code

As follow-up on practice *Review of GenAI-generated code*, student's code analysis can be used for assessment. This would fit learning objectives such as "students are able to evaluate the quality of code". Assessment categories could be accuracy and thoroughness.

Sources: (Becker et al., 2023; Kendon et al., 2023)

### Practice: Assessment of student reflection on GenAI usage for problem solving

Additional to students using GenAI to solve a programming problem, they also have to reflect on how they used it and how it helped them, which requires higher cognitive levels according to Bloom. This reflection is turned in and assessed.

Source: (Lau & Guo, 2023)

## References


Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023). Programming Is Hard - Or at Least It Used to Be. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 500–506. https://doi.org/10.1145/3545945.3569759

Cipriano, B. P., & Alves, P. (2023). GPT-3 vs Object Oriented Programming Assignments: An Experience Report. *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 61–67. https://doi.org/10.1145/3587102.3588814

Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B. A., & Reeves, B. N. (2024). Prompt Problems: A New Programming Exercise for the Generative AI Era. Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, 296–302. https://doi.org/10.1145/3626252.3630909

Kendon, T., Wu, L., & Aycock, J. (2023). AI-Generated Code Not Considered Harmful. *Proceedings of the 25th Western Canadian Conference on Computing Education*. https://doi.org/10.1145/3593342.3593349

Lau, S., & Guo, P. (2023). 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. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, 106–121. https://doi.org/10.1145/3568813.3600138

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1, 27–43.

https://doi.org/10.1145/3501385.3543957

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spence-Smith, J., & Schmidt, D. C. (2023). *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. https://doi.org/10.48550/arXiv.2302.11382