

FROM CODE WRITING TO CODE JUDGMENT: A BLOOM-BASED REDESIGN FOR LEARNING PROGRAMMING WITH GENAI

Cornelia Heinisch

IU International University of Applied Sciences (Germany)

Abstract

Generative AI (GenAI) tools such as ChatGPT and GitHub Copilot are reshaping introductory programming in higher education. Because these tools can already produce functional solutions for many standard programming tasks, the manual production of syntactically correct code is becoming a less sufficient indicator of programming competence. This conceptual paper examines how introductory programming tasks can be redesigned using the revised Bloom's taxonomy. It maps traditional task formats and GenAI-based redesigns to Bloom's cognitive levels, using examples from control structures and introductory object-oriented programming. The analysis highlights a shift in competence emphasis from lower levels such as Remember, Understand, and Apply toward higher levels such as Analyze, Evaluate, and Create. In this context, students increasingly need to interpret, compare, assess, and deliberately revise AI-generated code. The paper argues for the pedagogical integration of GenAI into programming courses and outlines implications for redesigning exercises and assessments to foreground reflective code judgment.

Keywords: *Generative AI, introductory programming, bloom's taxonomy, task redesign.*

1. Introduction and background

Generative AI (GenAI) tools such as ChatGPT and GitHub Copilot are increasingly used by students in introductory programming courses (CS1) to generate code for assignments, obtain explanations of code, and revise existing solutions. Students may use these tools even when they are not explicitly integrated into course design (Becker et al., 2023; Mahon, Mac Namee, & Becker, 2024). At the same time, GenAI tools are becoming more common in professional software development practice (Bird et al., 2022; Li, Arony, Awon, Damian, & Xu, 2024). As a result, programming education can no longer focus primarily on the manual production of syntactically correct code. Instead, students increasingly need to learn how to understand, examine, adapt, and critically evaluate generated solutions (Mahon, Mac Namee, & Becker, 2024).

In CS1, exercises and assessments have often emphasized the lower levels of the revised Bloom's taxonomy (Krathwohl, 2002), particularly Remember, Understand, and Apply (Fuller et al., 2007; Thompson, Luxton-Reilly, Whalley, Hu, & Robbins, 2008). These foundational levels remain important. However, if GenAI can already produce functional solutions for many tasks at these levels (Heinisch, 2025; Finnie-Ansley et al., 2022), the ability to write syntactically correct code loses value as a stand-alone indicator of programming competence.

Against this background, this paper addresses the following question: How should tasks and assessments in CS1 be redesigned so that, despite the availability and use of GenAI tools, students develop solid programming skills while also acquiring the additional competencies required for AI-supported software development in professional practice?

To address this question, the paper first maps traditional tasks from the areas of control structures and object-oriented programming to the levels of the revised Bloom's taxonomy: Remember, Understand, Apply, Analyze, Evaluate, and Create. The examples are presented in Java as an illustrative programming language, but the approach can be transferred to other languages. The paper then redesigns these tasks to explicitly incorporate the use of GenAI tools. By contrasting traditional and AI-based task formats, it highlights how the competence focus in introductory programming shifts from the production of code toward the judgment of code. In this context, GenAI is understood and pedagogically framed as a standard tool for software development, comparable to compilers and debuggers. The remainder of the paper presents this mapping, introduces the corresponding AI-based redesigns, and discusses their implications for CS1 teaching and assessment.

2. Analysis of traditional tasks and their classification within Bloom's taxonomy

Traditional tasks in CS1 primarily focus on foundational knowledge, code comprehension, and the syntactically correct application of familiar programming constructs. The revised Bloom's taxonomy provides a useful framework for structuring such tasks according to their cognitive demands. At the same time, the classification of programming tasks into individual Bloom levels must be understood as context-dependent. Fuller et al. (2007) argue that programming tasks can only be meaningfully classified with reference to the specific course materials and the prior knowledge already taught. If a task has already been addressed in class, working on it again is more appropriately assigned to the level of Remember. If, by contrast, it requires transferring a known pattern to a slightly modified case, it is more appropriately assigned to the level of Apply. Bloom classification is therefore not absolute, but depends on the didactic context. Table 1 illustrates how typical traditional CS1 tasks can be mapped to the levels of the revised Bloom's taxonomy.

Table 1. Typical traditional CS1 tasks across Bloom's taxonomy.

Bloom level	Typical traditional task
Remember	Name iteration and selection statements together with their keywords; define OOP concepts such as class and object.
Understand	Explain how <code>while</code> and <code>for</code> iteration statements differ; explain why the <code>this</code> reference is used in a constructor.
Apply	Write a loop to calculate a sum or average; implement a simple previously unknown class with instance variables, constructors, and getter and setter methods.
Analyze	Identify syntax or logic errors in given program code.
Evaluate	Compare two implementations in terms of performance, memory usage, or compliance with requirements.
Create	Design, assess, and implement a small class structure with inheritance, association, and aggregation relationships for a new problem; develop, implement, and assess an algorithm for a previously unknown problem.

In CS1, tasks at the levels of Remember, Understand, and Apply are particularly dominant. Their purpose is to reinforce fundamental knowledge of a programming language and to develop the ability to write syntactically correct program code for a given task. Typical examples include tasks such as: "Explain how the terms reference variable, class, object, `new` operator, and constructor are related" (Understand) or "Write a `for` loop to calculate the average of the values in an array" (Apply). Tasks at the levels of Analyze, Evaluate, and Create also occur, but they are considerably less frequent in introductory courses, especially in time-limited examinations.

This emphasis is didactically understandable, because competencies at higher Bloom levels build on foundational knowledge and elementary application skills. At the same time, the analysis shows that traditional task formats in CS1 primarily target those areas of competence that can now be particularly well supported by GenAI. For this reason, the next step is to examine how task formats can be didactically redesigned through the explicit inclusion of GenAI.

3. Redesign of ai-based tasks and their classification within Bloom's taxonomy

The availability of GenAI does not change the foundations of learning programming, but it does change the didactic function of many tasks in CS1. Foundational knowledge, conceptual understanding, and the ability to write syntactically correct program code remain necessary prerequisites for more advanced programming competencies. At the same time, under conditions of readily available GenAI, traditional task formats that primarily address the lower Bloom levels – Remember, Understand, and Apply – are becoming increasingly insufficient for making practically relevant competence visible. If generative AI can already provide functional solutions for many standard tasks, the focus of learning shifts from the mere production of code toward the understanding, checking, comparison, and deliberate revision of generated solutions.

AI-based tasks therefore differ from traditional tasks not simply because a GenAI tool is used. Rather, what is decisive is that interaction with the tool itself becomes part of the competence to be developed. Students must be able to classify, assess, and deliberately further develop AI-generated code, explanations, and tests. The following examples are based on typical learning and assessment tasks in an introductory programming course. Table 2 illustrates how traditional task formats can be redesigned through the explicit inclusion of GenAI.

Table 2. From traditional tasks to AI-based redesigns.

Bloom level	Traditional task	AI-based redesign
Remember	Name all iteration statements in the Java programming language.	Use a GenAI tool to generate five quiz questions on iteration statements in Java and answer them. Note the terms that you were not yet able to recall.
Understand	Explain the difference between while, for, and do-while loops.	Ask a GenAI tool to explain the differences between while, for, and do-while loops in Java and compare the explanation with the course materials. Use GenAI, if necessary, for additional explanations and examples.
Apply	Implement a method <code>avg()</code> that receives a numeric array as a parameter and returns the average of the values in the array.	Use GenAI to generate a solution for a method <code>avg()</code> that calculates the average of a given numeric array. Transfer this solution to slightly modified variants in which either an <code>int</code> array or a <code>float</code> array is processed, and execute the resulting variants.
Analyze	Find syntax and logic errors in a given faulty class.	Use a GenAI tool to generate Java code for a given task description. Analyze the generated code with regard to its conformity with the task description and identify missing, faulty, or superfluous elements.
Evaluate	Assess whether the data types chosen for a class are appropriate from a domain perspective.	Evaluate the data types chosen for the instance variables in an AI-generated class with regard to domain appropriateness, value range, and implications for functional correctness; justify which data types should be retained and which should be revised.
Create	Design a class hierarchy for managing different kinds of articles in an online shop.	Use GenAI for a first draft of a class hierarchy. Revise and complete it in accordance with object-oriented principles and document your central design decisions.

The following examples illustrate more concretely how AI support can shift tasks from the lower Bloom levels toward higher levels – that is, from code writing to code judgment.

3.1. Example 1: Control structures

A traditional CS1 task might be formulated as follows: “Implement a console program that reads two integers and an operator (+, −, *, /), performs the corresponding arithmetic operation on the operands, and outputs the result on the screen.”

This task lies primarily at the level of Apply, because familiar language constructs such as variables, input handling, and conditional branching are transferred to a typical standard task. The focus is on the correct production of program code.

An AI-based redesign, by contrast, might be formulated as follows: “Use a GenAI tool to generate two Java solutions for the task above: one variant using `else-if` and one variant using `switch`. Analyze both solutions in terms of correctness, readability, and extensibility. In particular, examine how invalid operators and division by zero are handled. Justify whether the `switch` variant or the `else-if` variant should be preferred for this task.”

Here, the cognitive demand shifts toward Analyze and Evaluate. Students no longer merely write a conditional structure themselves, but must assess generated solutions in terms of their fit to the task and their quality. The relevant competence therefore lies no longer primarily in writing a control structure for the first time, but in the well-founded analysis and evaluation of generated code.

3.2. Example 2: Object-oriented programming and testing

A traditional task might be formulated as follows: “Implement a class `UserProfile` with the attributes `name` and `age`, appropriate getter and setter methods, and a method `welcome()` that returns a greeting text.”

This task also lies mainly at the level of Apply, because a familiar schema of introductory object-oriented programming is reproduced: declaring attributes, implementing accessor methods, and formulating a simple instance method.

An AI-based redesign might be formulated as follows: “First, use GenAI to generate the program code for the class `UserProfile`. Then generate test cases in three different ways: (a) on the basis of the task description, (b) on the basis of the finished implementation, and (c) on the basis of a faulty implementation into which you have previously introduced a logic error in the method `welcome()`. Compare the generated test cases and execute them. Determine what must be taken into account when generating test cases with GenAI so that logic errors in the program code can actually be detected.”

Here, the focus shifts toward Analyze, Evaluate, and partly Create. The relevant competence no longer lies in writing a simple class skeleton for the first time, but in examining the quality of AI-generated implementations and AI-generated tests. Students must assess whether default values are tested correctly, whether getters and setters are actually covered, and whether the method `welcome()` is validated precisely against the expected output string. In this way, not only code, but also test quality itself becomes an object of judgment.

These examples show that, at the lower Bloom levels, GenAI can primarily support learning processes, whereas at the higher levels the well-founded examination, selection, and revision of generated artifacts themselves become the object of learning and assessment. The pedagogical added value thus arises not despite, but precisely because of, the deliberate integration of the tool.

4. Discussion and limitations

The comparison of traditional and AI-based tasks makes a fundamental shift in programming competence visible. In traditional CS1 formats, the ability to correctly produce small programs or code fragments is often treated as a central indicator of competence. Under conditions of widely available GenAI, however, this is no longer sufficient. If functional code can be generated quickly for many standard tasks, what becomes increasingly important is whether students can understand, contextualize, test, and deliberately revise that code.

For CS1 teaching, this suggests that the lower levels of the revised Bloom's taxonomy – Remember, Understand, and Apply – remain relevant, but partly take on a different didactic function. At these levels, GenAI can serve as a learning tool, for example by explaining concepts, varying practice tasks, or supporting students in working with syntax and basic programming structures. At the same time, the higher levels – Analyze, Evaluate, and Create – should be emphasized more systematically, as they capture competencies that are particularly important in AI-supported software development in professional practice (Bird et al., 2022; Li, Arony, Awon, Damian, & Xu, 2024).

For CS1 assessment, this implies that task formats should increasingly distinguish between learning with GenAI and demonstrating reflective judgment in working with GenAI. More relevant formats include tasks in which students compare generated solutions, identify weaknesses, justify decisions, and deliberately revise code. Accordingly, assessment should focus not only on the final solution, but also on the quality of the underlying analysis, evaluation, and justification.

From this perspective, both a blanket prohibition of AI tools and the unreflective continuation of traditional task formats appear pedagogically unconvincing. If GenAI is already part of students' learning environments and increasingly relevant in professional software development practice (Mahon, Mac Namee, & Becker, 2024; Bird et al., 2022; Li, Arony, Awon, Damian, & Xu, 2024), programming education should respond by redesigning tasks in ways that make higher-order competencies visible.

The paper also has clear limitations. It is intended as a conceptual contribution rather than an empirical effectiveness study. The proposed Bloom classifications and AI-based redesigns should therefore be understood as theoretically grounded examples rather than validated instructional interventions.

5. Conclusion and future work

This paper argues that generative AI is changing the didactic function of traditional programming tasks in CS1. Many classical tasks at the lower levels of the revised Bloom's taxonomy can now be supported or directly addressed by GenAI. As a result, the mere production of syntactically correct code is losing value as a stand-alone indicator of programming competence.

As a conceptual contribution, the paper shows that a Bloom-based redesign of tasks can make a shift in competence from code writing to code judgment visible. Using example tasks from control structures and introductory object-oriented programming, it illustrates how traditional tasks can be transformed into AI-based learning and assessment formats. Foundational knowledge and the ability to apply basic programming concepts remain essential, but competencies related to Analyze, Evaluate, and Create become increasingly important.

Future work should examine empirically how such AI-based task formats affect learning outcomes, assessment validity, and long-term competency development in CS1.

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: Educational opportunities and challenges of AI code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 500–506). New York, USA: ACM.
- Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., & Gazit, I. (2022). Taking flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools. *Queue*, 20(6), 35–57.
- Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The robots are coming: Exploring the implications of OpenAI Codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference* (pp. 10–19). New York, USA: ACM.
- Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., Lahtinen, E., Lewis, T. L., Thompson, D. M., Riedesel, C., & Thompson, E. (2007). Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bulletin*, 39(4), 152–170.
- Heinisch, C. (2025). Fostering computational thinking through AI-assisted teaching of multiple programming languages. In *EDULEARN25 Proceedings* (pp. 1489–1498). Palma, Spain: IATED.
- Krathwohl, D. R. (2002). A revision of Bloom’s taxonomy: An overview. *Theory Into Practice*, 41(4), 212–218.
- Li, Z. S., Arony, N. N., Awon, A. M., Damian, D., & Xu, B. (2024). AI tool use and adoption in software development by individuals and organizations: A grounded theory study. Retrieved March 12, 2026, from <https://arxiv.org/abs/2406.17325>
- Mahon, J., Mac Namee, B., & Becker, B. A. (2024). Guidelines for the evolving role of generative AI in introductory programming based on emerging practice. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (pp. 10–16). New York, USA: ACM.
- Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom’s taxonomy for CS assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education* (Vol. 78, pp. 155–161). Wollongong, Australia: Australian Computer Society, Inc.