A Conceptual Framework to Transform Coding Education in Times of Generative AI

Stefan Bente[1], Natasha Randall[2], Dennis Wäckerle[3]

Abstract: In light of the rising ubiquity and accessibility of AI tools, software engineering curricula must be adapted in response, particularly with regards to coding education. We propose an extension to Bloom's Revised Taxonomy, enhanced to a second dimension that represents the scope and complexity of coding tasks. The path that a computer science study program typically follows can be mapped onto these dimensions, and evaluated in the context of generative AI tool use, which falls into three core stages. AI should be initially banned, then made an explicit part of coding education, and finally becomes an implicit part of the coding practices of students.

Keywords: Coding; ChatGPT; Software Engineering; Education; Bloom's Revised Taxonomy

1 Introduction

The rise of accessible generative AI tools such as ChatGPT and GitHub Copilot - capable of producing large amounts of text and code at the click of a button - has led to fears that students will simply use AI to cheat on assignments, at the expense of learning [SKM23]. These fears have been validated by research from Geng et al. suggesting that ChatGPT is capable of passing university level programming courses [Ge23b] [Ma23]. On the other hand, Krüger and Gref demonstrate how ChatGPT's mathematical limitations can significantly reduce its effectiveness in a computer science degree program [KG23], and a new study by Jimenez et al. reveals how ChatGPT completely fails to handle real world software engineering tasks [Ji23]. Nevertheless, many developers increasingly believe that traditional programming is "headed for extinction" [We22].

A considered approach to generative AI in educational settings will therefore be key. But how exactly should SE curricula be adapted, specifically with regards to coding education, i.e. teaching programming skills? There are a number of recent publications on this topic ([Ab23], [Li23], [DB23], [Ja23], or [Ge23a], just to name a few). However, what is lacking is an easy-to-understand mental model "where, when, and how" to integrate generative AI into an SE curriculum. In section 2 we attempt to close this gap by proposing a simple framework to assess coding education. We enhance Bloom's Revised Taxonomy [AKB01] to a second dimension, namely the "scope" of coding tasks, which encapsulates aspects

[1] TH Köln, Fakultät für Informatik und Ingenieurwissenschaften, stefan.bente@th-koeln.de
[2] TH Köln, Fakultät für Informations- und Kommunikationswissenschaften, natasha.randall.uni@gmail.com
[3] TH Köln, Fakultät für Informatik und Ingenieurwissenschaften, dennis.waeckerle@smail.th-koeln.de

doi:10.18420/seuh2024_08

relating to the complexity of a task. In section 3 we then validate this conceptual framework by analysing the Computer Science Bachelor at TH Köln, with a study that examines the extent to which ChatGPT is able to solve curricular assignments focusing on coding skills.

The combination of these two dimensions allows us to draw the "path" a computer science study program typically follows. This path can be segmented into three stages regarding how to embrace AI tools in coding, as described in section 4. Initially, we recommend to *ban AI* in the very first stages of programming education, to allow students to develop basic hands-on coding knowledge. In the second stage, we propose to make the use of AI tools an *explicit part of the coding education*, to let students develop a sense for the strengths and weaknesses of AI tools. Finally, in that last stage, AI tools should be an *implicit, unconstrained part of coding practice*, even up to their availability in exams - just the situation the students will face later in their professional context. Finally, we discuss open issues and further research of our proposal in section 5.

2 A Conceptual Framework to Describe Coding Education

A fundamental concept underlying effective coding education is *Active Learning* [HRL20, p. 23ff], which simplifies to the idea that true understanding comes through hands-on experience. As novices gradually progress, they ascend to higher cognitive levels, such as analyzing requirements and domains, and assessing existing code and technologies. Bloom's Revised Taxonomy [AKB01, p. 28f.] provides a framework for understanding such levels, encompassing six distinct stages in a "somewhat hierarchical"[4] framework, ranging from lower-order thinking skills like remembering and understanding to higher-order skills such as applying, analyzing, evaluating, and creating. It provides educators with a structure to design learning activities and assessments that promote progressively deeper cognitive engagement.

For beginners in coding, the journey typically begins with small code snippets, aligning with the lower cognitive levels in Bloom's Taxonomy: remembering, understanding, and applying concepts. But how to progress from here? The reality as an IT professional means dealing with large software systems, with up to millions of Lines of Code (LoC), and a multitude of technologies, architecture styles, and code patterns. This is a long way from the 10-100 LoC coding tasks for novices. This journey requires *two dimensions* to be described properly: On the one side, as students advance, they should be able to analyse requirements to turn them into code (Bloom level 4), read, understand, and assess the quality of code (level 5), and even think of new, innovative coding patterns or technologies (level 6). This dimension of students' learning journey through IT problem solving is well covered by the Bloom taxonomy.

[4] The question if the Revised Bloom's Taxonomy forms a *cumulative hierarchy* has more to it than meets the eye. The authors themselves give a differentiated "yes-and-no" answer that is worth reading [AKB01, p. 267f.].

On the other side, students' coding assignments should evolve to encompass the development of full-fledged programs with 1000 - 10.000 LoC. Structural considerations like software architecture become more and more relevant on the way: A combination of perfectly well designed small code snippets may ultimately end up becoming a horrifying "big ball of mud", if such aspects are disregarded. This dimension of "learning to code" is not covered by Bloom's Revised Taxonomy[5]. We therefore propose a second dimension that we call *scope*. Scope encompasses various aspects that collectively contribute to a task's complexity, including (but not limited to) size, number of classes, technology layers (like persistence annotations for an ORM layer), architecture styles (layered architecture, hexagonal, ...), Clean Code rules, code quality metrics (number of dependency cycles, cyclomatic complexity, ...), performance metrics, etc. For the sake of simplicity, within the context of this paper, we assume that the average size of the solutions' code base can serve as a valid indicator for scope, and that larger coding exercises will (have to) cover these additional aspects to provide a useful learning experience to students.[6]

Figure 1 depicts these two dimensions. The coding-related modules of an SE curriculum need to follow a structured path from the bottom left corner (low cognitive level, small code snippets) to intricate programming challenges with "Evaluate" level and large software systems in the top right corner. The stages that students traverse in their curriculum appear as quadrants in a 2-dimensional matrix.

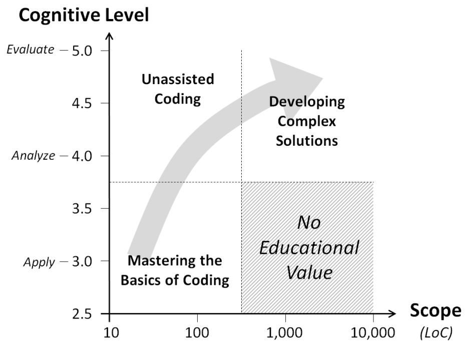

Fig. 1: Coding Education as a 2-dim Matrix

Initially, the students need to *master the basics of coding*, mainly by applying (Bloom level 3) programming language syntax rules to small tasks (bottom-left). In the second stage, the students move on to *unassisted coding* (top-left), with the ability to analyze (level 4) requirements and domains, and evaluate (level 5) existing code for refactoring, to optimize it for quality attributes like maintainability or performance. In the last stage (top-right), students will slowly broaden the scope and use their skills to *develop complex solutions*. The bottom-right quadrant has no educational value, as it would just mean applying the same detailed instructions over and over to a very large task, without ever gaining a deeper understanding.

The y-axis in figure 1 employs decimal values for Bloom levels, not discrete numbers. If a

[5] Anderson and Krathwohl do actually propose a second dimension of their taxonomy, called the "knowledge dimension" [AKB01, p. 28ff.]. It distinguishes between factual, conceptual, procedural, and meta-cognitive knowledge. This is of course a relevant distinction, but doesn't really help us here in describing how coding exercises need to evolve through the curriculum.

[6] This assumption will require further research, see also the outlook in section 5 of this paper.

coding-related module's practical e.g. consists of 20 assignments, we assessed the highest required Bloom level for each of them, then calculated the average, resulting in a decimal number. Likewise, the LoC is determined by averaging assignment LoC. Data points were generated using either existing student solutions or newly created AI solutions. The exact data is detailed in table 1. In addition, the range of values in the chart is deliberately restricted between 2.5 and 5. The underlying assumption is that no module teaching coding (or anything else, for that matter) will stay mainly on Bloom level 1 or 2 - this would mean that students only learn by heart or explain concepts, but never actually *apply* any of the competences they acquire. A Bloom level higher than 5, on the other hand, will be found in the realm of a Bachelor thesis, but rarely (as an average over the whole semester) in a regular course. The evaluation of a concrete study program, as described in the section 3, supports this assumption.

ID	**Module Name**	**Sem.**	**Description**	**Coding ECTS**	**Avg. Bloom**	**Avg. LoC**
AP1	Algorithms and Programming I	1	Programming exercises in C and Java	8	3.2	35
AP2	Algorithms and Programming II	2	Programming exercises and patterns in Kotlin	7	3.5	117
ALG	Algorithmics	3	Algorithms (sorting, graphs, ...)	1.7	4.0	263
DB1	Databases I	3	ERM and basic SQL	2.3	3.4	43
PoP	Paradigms of Programming	3	OO vs. functional vs. logic programming	5	3.4	53
SE1	Software Engineering I	3	Domain modelling in Java and Spring	1.6	3.8	51
DB2	Databases II	4	Advanced SQL in client-server systems	5	4.0	1507
SE2	Software Engineering II	4	Complex system according to DDD in Java	4.1	4.4	1937
AI	Artificial Intelligence	5	Practical AI application	5	4.7	653
CSP*	Computer Science Project	5	Practical programming project	10*	>=4.0*	>= 5000*
PP*	Practical Project	6	Project intended as base for BA thesis	15*	>=5.0*	>= 10000*
BA*	Bachelor Thesis	6	Scientific thesis, sometimes on coding topics	12*	>=5.0*	>= 10000*

Tab. 1: (Potentially) Coding-related Modules in Computer Science Bachelor at TH Köln

3 A Brief Case Study: Computer Science Bachelor at TH Köln

In sample a case study, we analyzed the Computer Science Bachelor program at TH Köln. The coding-related parts of this program amount to 47 - 75 ECTS[7].

We evaluated the mandatory coding-related courses regarding their average code size and Bloom level; the results are shown in table 1 (German module names have been translated to English). The numbers in table 1 were obtained by analyzing sample student solutions, or reference solutions provided by the supervisors, where available[8]. The modules marked with an asterisk are only potentially coding-related. Their numbers represent an "optimistic" value obtained by averaging selected samples with a high degree of coding. Figure 2 shows the modules in table 1 positioned in the 2-dimensional Bloom grid introduced in section 2, marked by their ID and semester in which they should take place according to the study plan. The potentially coding-related modules are depicted as dotted circles, to visualize the high degree of uncertainty.

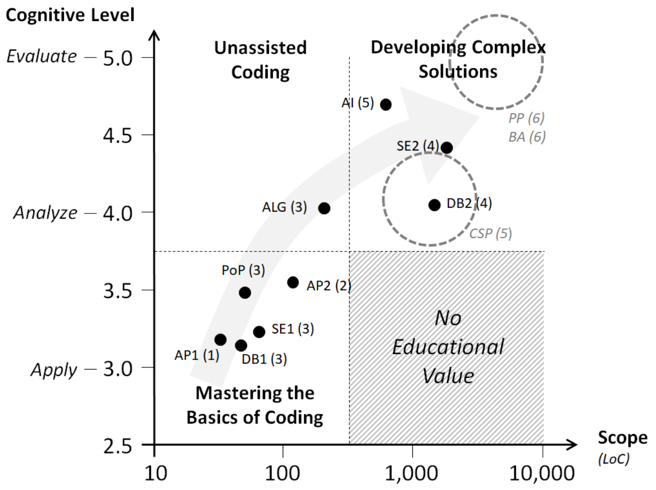

Fig. 2: Positioning of Modules in 2-dim Bloom Matrix

3.1 ChatGPT and Current Assignments

To ascertain the impact of AI tools on coding exercises, we attempted to solve a subset of the modules' assignments in table 1 (AP1, PoP, and SE2) using ChatGPT 3.5 and 4.0. We used an *uneducated approach*, meaning that only the tasks' descriptions, outputs generated by the compiler, test outputs, and bugs could be provided to ChatGPT. No other input or manual coding was allowed.

[7] Elective courses, Bachelor thesis, and various project formats can contain a varying amount of coding-related content, hence the range.

[8] The way LoC are counted is by looking at the *context* in which the exercise happens.
In AP1 and PoP, for instance, there is a new (relatively small) context in every exercise, therefore each context is relatively small. In AP2 students sometimes have to extend the work from a previous exercise. In SE2, iteration on the same code base is a core principle, therefore the LoC numbers are higher.

The modules that we mainly focused on during this research were AP1 and SE2. AP1 is an introductory module, teaching students the very fundamentals of programming in C and Java. It is a mix of procedural and object-oriented assignments, completed throughout the entire semester. All of these tasks are elementary, do not involve any complex maths or logic, and most can be completed in a single file or class.

SE2 on the other hand, is a more advanced module teaching students how to develop larger scale applications using Domain-Driven Design (DDD) and Spring Boot. SE2 has fewer but larger exercises in Java, designed as milestones that all iterate on the same application. The tasks are free-form, allowing the students to build the application in any way they see fit, as long as it implements the necessary interfaces and complies with the regression and architectural tests. These applications always span multiple classes which are in the double digits, and thus have an entirely different scope than the coding exercises in the other modules. The complexity and the free-form approach requires a higher level on the Bloom Taxonomy, with most tasks being level 4 and some even reaching level 5.

It was immediately obvious that generative AI tools excel in generating programs small in scope. AP1 (1st semester) could essentially be solved in its entirety with no input other than the task description, within one hour. The same result was achieved with PoP (3rd semester), where ChatGPT was able to solve the given problems mostly on the first try.

Looking at SE2, when attempting to solve the milestone 0 of the summer semester of 23, which required the implementation of a simple B2B webshop, the entire milestone could be solved using no human written code, despite the large scope. However, the AI started struggling in the next milestone once architectural best practices started being enforced, which the AI seemingly was unable to solve on its own, especially regarding resolving cyclic dependencies. Furthermore, some of its default implementations were violating DDD principles, which would cause the code to fail tests in later milestones.

Meanwhile, the AI could not solve milestone 0 of the summer semester of 21, where a program should be implemented which simulates the movement of robots on a grid of cells containing barriers between some cells. The AI was incapable of implementing the necessary logic for the barriers, meaning that it was failing the automated tests, despite the scope of the task being slightly smaller than the tasks of the summer semester of 23. While this result seems surprising at first sight, it also confirms the AI tool's well-known struggle with maths and similar algorithmic challenges [Zh23].

Our observations match the results from numerous other studies on ChatGPT and curricular coding assignments. "Our findings show that ChatGPT is effective for easy and popular coding problems but is less reliable for harder and less popular problems" write Kuhail et al. [Ku23]; [KG23], [DB23], and [Ge23a] come to a similar conclusion. Single file programs with no additional challenges can easily be solved by students using an "uneducated" approach with ChatGPT, i.e. without ever trying to understand the assignment. The capabilities mentioned above are further enhanced when using widely known technologies and programming languages. With more complex assignments, in the top-right *Develop Complex solutions* quadrant, ChatGPT can still be an extremely helpful tool, but nevertheless requires some manual interventions. Students will have a hard time solving such assignments with a purely uneducated approach.

4 Recommendations Regarding AI Tool Usage

Based on the capabilities of AI tools, new ways of teaching and learning must be developed. Jacques [Ja23] states that "[...] at the very least, we have a responsibility to prepare our students for this transition. Specifically, we need to consider how to develop good programmers while still acknowledging and engaging with these new tools for programming."

Combining the conceptual grid from section 2 and the results from our case study in section 3 (and many others), we propose three stages in dealing with AI in SE curricula. This is depicted in figure 3.

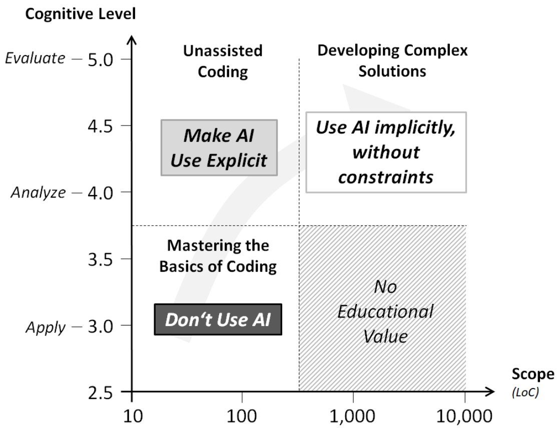

Fig. 3: Three Stages for AI Usage in SE Curricula

4.1 Stage 1 (Mastering the Basics of Coding): Don't Use AI

In the first stage of learning to code, it seems advisable to ban the use of AI tools during exercises, practicals, and exams. In this phase, students learn by simply applying syntax definitions, and by following quite detailed instructions. AI tools like ChatGPT offer an all too easy way out of this, as demonstrated in section 3.1. In addition, an unpublished research paper [DOH22] from the 2022 "Coding Excellence" course in Master Digital Sciences showed that beginners actually profit more from simpler programming tools, like syntactical code completion - rather than full-fledged AI tools like ChatGPT, GitHub Copilot, Tabnine etc., which are seemingly prone to creating a cognitive overload for novices.

Exams in coding-related courses should be digital and based on real-life development environments, if possible (following the "Constructive Alignment" principle [Bi03]). If the university has a large enough computer lab, then exams should take place there, as it is relatively easy to block access to unwanted tools in such a controlled environment.

4.2 Stage 2 (Unassisted Coding): Make AI Use Explicit

In this second stage of their “coding journey”, students have acquired the fundamental knowledge about how to write code - from a *computational thinking* approach to problem solving (at least for isolated tasks) to a basic familiarity with the syntax of one or two main programming languages, and they are capable of developing small scale programs on their own. This stage therefore focuses on deepening their understanding of the numerous aspects associated with the coding lifecycle, like domain exploration, algorithms, data structures, code patterns, quality attributes (e.g. through established rulesets like Clean Code), testing, debugging, source code management, deployment, etc. In addition, many other functional and non-functional properties, depending on the task at hand, might have to be checked, e.g. algorithmic suitability, to avoid performance problems with large data sets.

ChatGPT (and comparable AI tools) are quite proficient as a “Swiss army knife” in complex coding tasks - when used in an educated fashion. Their capabilities extend beyond single file programs, to ones featuring multiple classes and methods. A strong feature of ChatGPT is its ability to “scaffold” an application - the “hello world” creation, if you like - in an unfamiliar language or framework, like e.g. Spring Boot. This effectively reduces the frustrations of beginners to programming when learning new, complex technologies, since it can free them from spending a couple of days just getting the syntax right.

When problems arise, ChatGPT has also proven to be quite capable of identifying bugs and fixing them. This includes the ability to analyse stack traces - especially useful when working with a framework like Spring Boot[9]. Furthermore, the refactoring of existing code is also something that ChatGPT is quite proficient in. In one case, when asked to refactor an existing class to use some specific Clean Code rules in an SE2 assignment, the AI’s results were very close to perfection, only requiring some minor manual adjustments.

To ensure an *educated use* of AI tools, assignments in the “Unassisted Coding” phase need to train students in *AI Literacy*. This is defined by Long and Magerko [LM20] as “a set of competencies that enables individuals to critically evaluate AI technologies [...]”. The authors then describe 16 core competencies associated with AI Literacy. For the focus area of this paper (using AI in coding education), competency 5 (AI’s Strengths & Weaknesses) is especially important.

The definition by Long and Magerko sums up the key goal for actively embracing AI tools in SE curricula during this stage: “Identify problem types that AI excels at and problems that are more challenging for AI. Use this information to determine when it is appropriate to use AI and when to leverage human skills”. Regarding ChatGPT as a coding tool, this means that students can read and understand the generated code, and assess if it really fits to the task description given in the prompt. This competence fits to the “Unassisted Coding”

[9] Spring Boot is the de-facto industry standard for Java-based client-server applications (and should therefore be taught in SE curricula), but drives programming beginners mad with its infamous “stack traces from hell” [Nu12].

phase, where assignments need to include the *Evaluate* (5) Bloom level, to check code for the aforementioned aspects and quality attributes.

This would also include training the students in recognizing ChatGPT's known weaknesses. As our case study (and other literature like [KG23]) shows, AI tools are (currently) error-prone when solving maths and logic assignments. A further weakness is the tendency of AI to add more complexity and to rewrite existing code in strange ways, when being asked to iterate on it. This tendency leads to harder to understand and more complex code, without any obvious advantages. Another weakness we observed is that working with AI and less well-known technologies yields worse results, with the AI having a greater tendency to "hallucinate" methods that don't exist. Students need to actively use ChatGPT (and other AI tools) to gain a clear understanding of both their potential and their weaknesses.

This goal can be achieved by making the use of ChatGPT an *explicit part of the assignments*. There are manifold creative ways to include AI Literacy in the learning outcome of coding-focused modules. [Ge23a][10], [DB23], and [Ja23] make proposals in that direction (see the selection in table 2). Any interactions between ChatGPT and the student should also be documented and reflected upon, as a mandatory part of the solution.

Method	Source
Ask the students to let ChatGPT generate code, and then analyze and explain it	[Ja23]
Let students implement a solution using an alternative approach to ChatGPT's solution	[Ja23]
Ask students for a different representation for ChatGPT's solution, e.g. for a given data structure	[Ja23]
Let the students use ChatGPT for self assessment of their code, and ask them to reflect on ChatGPT's improvement proposals	[DB23]
Let ChatGPT create coding tasks for students, and ask students to solve them	[DB23] [Ge23a]
Ask ChatGPT to create a faulty solution to a given problem, and then ask the students correct it	[Ge23a]
Let ChatGPT act as a teacher, explaining concepts and solutions	[Ge23a]
Switch roles between students and teacher: Give a reference solution to the students and instruct them to repeatedly prompt ChatGPT until it has generated a valid solution	[Ge23a]

Tab. 2: Innovative ways in which ChatGPT and similar tools can be used in SE education

As in stage 1, exams in coding-related courses in the "Unassisted Coding" phase need to be digital. They probably have to be two-phased: In phase one, access to ChatGPT needs to be open, so that students can work on assignments with explicit instructions to use AI tools, and document their interactions. In the second phase, with assignments focusing on other competencies, access must be blocked, since the scope of coding tasks is still

[10] This paper was written based on the results of the same teaching-research project as in this paper. The "DBS" module described there is the successor of "DB2" covered in this paper, in a subsequent examination regulation of the same study program.

relatively small, and ChatGPT would allow for an "uneducated" solution. A digital exam in a university's computer lab would allow such a configuration switch with relative ease.

4.3 Stage 3 (Developing Complex Solutions): Use AI implicitly, without constraints

The third and final stage allows the students unrestricted access to AI tools. During this stage, the scope of the tasks should be vast, to the point where using an "uneducated" approach would not be viable. This could be either done in large-scale practicals such as the ones in SE2 or DB2, or in large projects as found in the Computer Science Project (CSP), the Practical Project (PP) or Bachelor Thesis (BA). At this point, it would be up to the students to determine for which problems they would utilize the AI tools, as they should have learned in the previous stage how and where these tools perform best.

Exams are a great challenge in this phase. They need to be digital - as in the previous stages - and it would be natural to allow the students free access to AI tools during the exam. However, current exams are tailored towards a very limited time frame and assume no access to AI; these exams' small scope would therefore make an "uneducated" approach possible, and thus would fail to assess the students' true skills and knowledge. To still properly evaluate students when allowing them unrestricted use of AI tools, this scope needs to be enhanced. Unfortunately, this entails many changes, compared to the current exam design.

- With (far) larger assignments, manual corrections might not be viable anymore, due to the sheer lack of (human supervisor) resources. The correction therefore needs to be automized to some extent - quite a challenge when testing functional and non-functional properties of written code.
- For fairness reasons, the students need to have free access to the same industry-standard AI tools they use outside of exams. Most AI tools are not free. Some (like GitHub Copilot) offer a free, or at least inexpensive, academic license. Others, like ChatGPT 4.0, do not even (at the time this paper is written) offer such a licensing model.
- An alternative would be to abort written exams altogether in this phase, and require project submissions instead. This, however, would greatly increase the workload on the supervisors, due to the large scope of the code they need to evaluate.

This situation requires further discussion and research. However, in the spirit of Constructive Alignment [Bi03] as a guiding didactical principle, the exam needs to reflect the learning outcome, which in turn should match the situation the students will face later in their professional context. So it seems we just need to solve this, one way or another.

5 Conclusion and Outlook

Based on our research findings, we recommend that SE curricula should be transformed, following the three stages proposed in figure 3. Modules teaching the fundamentals of programming should forbid AI tools entirely, while modules falling into stage 2 (unassisted coding) and stage 3 (developing complex solutions) need adjustment. Special attention should be given to the modules in stage 2, as these require the explicit usage of AI tools and thus the most work. Modules in stage 3 need to have a vast scope. Therefore, they will then be less affected by "uneducated" AI tool use. AI tools need to be available during the students' examinations.

One limitation of our proposed conceptual framework is the simplified use of LoC to represent the scope dimension, as this does not fully represent the higher complexity of more advanced exercises. It also does not account for the shortcomings and strengths of AI tools as demonstrated with the SE2 practical in the summer semester of 2021 (see section 3.1). Further research should therefore explore other suitable attributes to represent scope. In addition, further validation of the model is required; systematic studies of coding education within other universities' SE curricula, would help to verify its wider applicability and generalisability.

It should also be further explored how ChatGPT and other such tools could be included in creative ways within coding education - and the same applies to exam design in the age of AI tools. Finally, university teachers, researchers, and their professional organizations need to lobby for free educational licences of ChatGPT 4.0 and other industry standard AI tools. This is required to provide a level playing field for all students.

Ultimately, we feel that the glass is half full, rather than half empty, and that AI tools will be a great enrichment for Software Engineering education in general - if we manage to adapt our curricula accordingly.

Bibliography

[Ab23] Abdelfattah, Aly Maher; Ali, Nabila Ahmed; Elaziz, Mohamed Abd; Ammar, Hany H: Roadmap for Software Engineering Education using ChatGPT. In: 2023 International Conference on Artificial Intelligence Science and Applications in Industry and Society (CAISAIS). p. 1–6, September 2023.

[AKB01] Anderson, Lorin W.; Kratwohl, David R.; Bloom, Benjamin Samuel: A Taxonomy for Learning, Teaching and Assessing: a Revision of Bloom's Taxonomy. Longman, New York, 2001.

[Bi03] Biggs, John: Aligning teaching for constructing learning. Higher Education Academy, pp. 1–4, 2003.

[DB23] Daun, Marian; Brings, Jennifer: How ChatGPT Will Change Software Engineering Education. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. pp. 110–116, 2023.

[DOH22] Dejean, Alexandre; Oedingen, Marc; Hammer, Maximilian: Artificial Intelligence-supported code completion. Research paper in module 'Coding Excellence' in Master Digital Sciences, 2022.

[Ge23a] Geisel, Victoria; Schindler, Christian; Stein, Nils; Bente, Stefan: Lernräume unter Verwendung von generativen Sprachmodellen. paper submitted to SEUH 2024, 2023.

[Ge23b] Geng, Chuqin; Yihan, Zhang; Pientka, Brigitte; Si, Xujie: Can ChatGPT Pass An Introductory Level Functional Language Programming Course? arXiv preprint arXiv:2305.02230, 2023.

[HRL20] Hazzan, Orit; Ragonis, Noa; Lapidot, Tami: Guide to Teaching Computer Science: An Activity-Based Approach. Springer Nature, 2020.

[Ja23] Jacques, Lorraine: Teaching CS-101 at the Dawn of ChatGPT. ACM Inroads, 14(2):40–46, may 2023.

[Ji23] Jimenez, Carlos E; Yang, John; Wettig, Alexander; Yao, Shunyu; Pei, Kexin; Press, Ofir; Narasimhan, Karthik: SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv preprint arXiv:2310.06770, 2023.

[KG23] Krüger, Tim; Gref, Michael: Performance of Large Language Models in a Computer Science Degree Program. arXiv preprint arXiv:2308.02432, 2023.

[Ku23] Kuhail, Mohammad Amin; Mathew, Sujith Samuel; Khalil, Ashraf; Berengueres, Jose; Shah, Syed Jawad: "Will I Be Replaced?" Assessing Chatgpt's Effect on Software Development and Programmer Perceptions of Ai Tools. SSRN, 2023.

[Li23] Li, Yihao; Xu, Jialong; Zhu, Yinghua; Liu, Huashuo; Liu, Pan: The Impact of ChatGPT on Software Engineering Education: A Quick Peek. In: 2023 10th International Conference on Dependable Systems and Their Applications (DSA). p. 595–596, August 2023.

[LM20] Long, Duri; Magerko, Brian: What is AI Literacy? Competencies and Design Considerations. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. CHI '20, Association for Computing Machinery, New York, NY, USA, p. 1–16, 2020.

[Ma23] Malinka, Kamil; Peresíni, Martin; Firc, Anton; Hujnak, Ondrej; Janus, Filip: On the educational impact of ChatGPT: Is Artificial Intelligence ready to obtain a university degree? In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1. pp. 47–53, 2023.

[Nu12] Nurkiewicz, Tomasz: , Filtering the Stack Trace From Hell, 2012.

[SKM23] Sullivan, Miriam; Kelly, Andrew; McLaughlan, Paul: ChatGPT in higher education: Considerations for academic integrity and student learning. 2023.

[We22] Welsh, Matt: The end of programming. Communications of the ACM, 66(1):34–35, 2022.

[Zh23] Zhou, Aojun; Wang, Ke; Lu, Zimu; Shi, Weikang; Luo, Sichun; Qin, Zipeng; Lu, Shaoqing; Jia, Anya; Song, Linqi; Zhan, Mingjie; Li, Hongsheng: Solving Challenging Math Word Problems Using GPT-4 Code Interpreter with Code-based Self-Verification. arXiv e-prints, 2023.