A.1 Teaching Computer Science Fundamentals to Business Students: A SoTL Experience Report on How to Increase Student Engagement via Practical Programming Tasks and AI

Ronny Seiger
School of Computer Science, University of St.Gallen, Switzerland

1 Introduction

The ongoing hype revolving around large language models and AI-based chatbots draws a lot of attention towards computer science (CS) education at universities (MacNeil et al., 2023; Sledgianowski et al., 2017). Increasing numbers of course offerings related to machine learning and data science targeted at CS and non-CS students can be observed worldwide. While these courses put focus on concepts and applications of artificial intelligence (AI), the interest in more fundamental CS concepts is declining, especially among students not majoring in CS.

This paper investigates an elective course teaching Fundamentals of Computer Science (FCS) that covers topics such as data structures and algorithms following Sedgewick and Wayne (2011) to students majoring in programs related to business administration. The course was established as one of the university's first CS offerings in 2018 to satisfy an increasing demand of topics related to digitization, data science, and information systems—emerging at a university that was primarily focused on business and law studies. The course was initially designed to follow a traditional front-of-class teaching to accommodate 50+ students. However, the course has been suffering from declining student numbers for several years, resulting in approx. 10 active students each year. To adapt to these numbers and counteract decreasing student engagement, we decided to introduce new in-class programming exercises to make the course more interactive. We report on the results of this *teaching innovation project* focusing on the research question:

How should practical programming tasks be designed to increase student engagement in fundamental computer science lectures for non-CS students?

We deem the course offering fundamental CS education to non-CS students to be very relevant as with its learning objectives guiding the course design (Biggs & Tang, 2010), it aims to provide insights into basic algorithms, data structures, software, and the programming of computers, which build the foundations of modern AI applications and information systems (Liebenberg et al., 2015; von Wangenheim & da Silva, 2009; Sanders & Mueller, 2000). The course puts a strong focus on *Conceptual* and *Procedural* knowledge as classified in Krathwohl (2002).

2 Methods

The goal of this project is to transition the FCS course from classical teaching towards a learning-oriented setting (Cerbin, 2018; Barr & Tagg, 1995) by introducing new ways for students to engage with CS topics. The focus is on *Student Engagement* comprising the students' interactions with materials, lecturers, and their peers as defined by Bigatel and Williams (2015).

We follow the *Scholarship of Teaching and Learning* (SoTL) research design method of educators studying their own teaching and students' learning to improve learning and share insights (Siegel, 2023; McKinney, 2010). This article serves as *innovation report* (Ludwig et al., 2018) to communicate the findings of a teaching innovation project conducted in a CAS (Certificate of Advanced Studies) program on higher education and teaching in fall semesters 2022 and 2023. The *Hopscotch 4-SoTL* toolbox (Steiner, 2023) serves as framework. First, we informally analyze student engagement in the FCS course *as-is*. Then, we propose to introduce new in-class programming exercises to intervene and counteract the decreased student engagement, and we analyze effect of these changes based on the lenses of critically reflective practice (Brookfield, 2017): a) our autobiography as a lecturer, b) our student's eyes, and c) our colleagues' experience.

2.1 Analyzing the As-is Situation

Due to the short project runtime of 18 months, the lecture only taking place on a yearly basis, and the unavailability of students from previous years, we base the analysis of the *as-is* situation primarily on the personal experience and observations

of this paper's author who held the lecture for the first time in 2022 using teaching materials supplied by its predecessor. In this course instance, the lecturer documented its observations made during a session and reflected retrospectively (*Reflection-on-action*) on the student interactions and discussions (Schon & DeSanctis, 1986). Additionally, we collect evidence via retrospective expert interviews with the lecturer who held the lectures and supplied the materials before 2022, and a lecturer holding other parts of the course. We can also rely on observations made by an expert from the university's center of learning and teaching as part of a shadowing session.

2.2 Introducing New In-class Exercises

We decided for the lecture in 2023 to introduce new in-class programming exercises. With programming tasks, students can directly test their understanding of new concepts and apply them in a practical context (Borowczak & Burrows, 2019). As the lectures and exercises are already programming focused with code examples provided in Python, we can assume that students are familiar with programming. An overview of the lectures, topics, and new exercises can be found in Table 1.

Table 1. Lectures and new in-class exercise contents

Lecture	Topic	New In-class Exercises
L1	Data Structures	Data cleaning operations String manipulation Using Lists and Dictionaries
L2	Classes & Objects	Using and extending provided classes Implementing own classes
L3	Complexity, Search Algorithms	Searching with Linear Search and Binary Search Profiling of algorithms (#comparisons, time)
L4	Sorting Algorithms	Sorting with Insertion Sort and Merge Sort Profiling of sorting algorithms (#comparisons, #exchanges, time)
L5	Graphs	Creating and visualizing graphs from data Computing graph metrics (node degrees, shortest paths)

To make the programming tasks more tangible and relevant, we used datasets from a *smart factory* as a running example (Seiger et al., 2022). These types of Industry 4.0 learning factories have proven their value in research (Malburg et al., 2020) and are highly appreciated by students for the hands-on experience. Depending on the type of task, students were provided with a specific dataset for data processing and analysis (cf. Table 1). The tasks and datasets are openly available in Seiger (2024).

Figure 1 shows an exemplary task related to lecture L2 on *Classes & Objects*. Students are provided with a brief introduction followed by specific tasks around the dataset. Figure 2 shows an exercise related to L4 dealing with *Sorting Algorithms*. Here we can see that additional hints and expected outputs of solutions are provided.

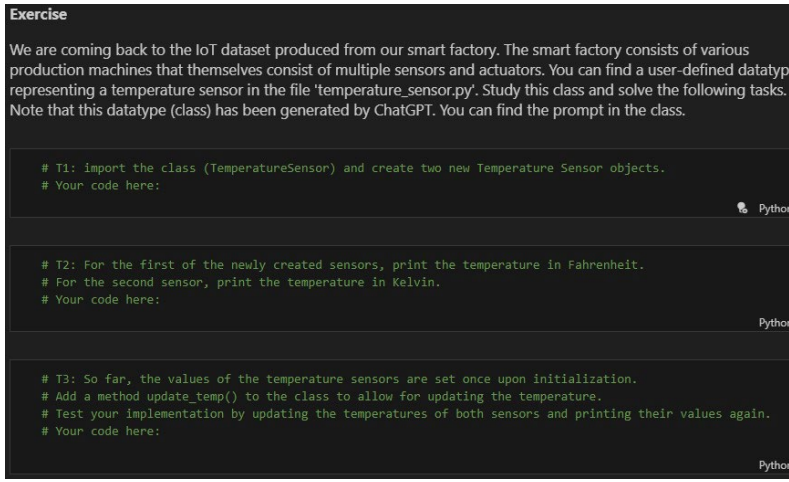

Figure 1: Screenshot of an in-class exercise for L2

The lecture materials, code examples, and new in-class exercises were integrated in *Jupyter* notebooks. Per 90-minutes lecture, there were two to three exercises, after a content-related unit from the lecturer ended. Each task was planned to be 5-10 minutes of students working, followed by brief discussions of solutions with the

lecturer and peers. Students were encouraged to follow a *pair programming* approach (Hanks et al., 2011).

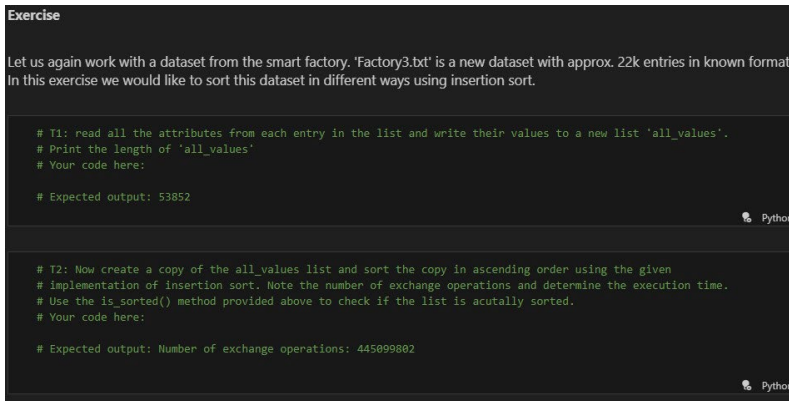

Figure 2. Screenshot of an in-class exercise for L4

To make the exercises more engaging, we encouraged the students to freely use AI (ChatGPT v3.5) to assist with the programming tasks. Based on the experience collected from the students after working on the tasks, we hypothesize that AI can be mostly helpful to support novices with simple programming tasks, e.g., by explaining how:

- to replace all occurrences of a character in a string in L1,
- to reverse sort a list in L1,
- the code for a sorting or search algorithm in L3 & L4 looks like and works,
- to use the *networkx* library for drawing a graph in L5.

We also asked the students to prompt ChatGPT to generate entire Python classes in L2 from a provided textual description. This worked well if the prompts are precise and detailed enough and the prompter correctly encodes its software design knowledge into the prompts. To challenge the chatbot, we asked it to generate a class for a temperature sensor with additional functionalities (e.g., to create invoices and ship items), which was done correctly by adding new methods to the class.

However, this violates a basic software design principle know as *Single Responsibility Principle* (Martin, 2017). From this and similar experiences we conclude that AI can help boost productivity when used for simple and repetitive coding tasks and for explaining code (Becker et al., 2023), but it may be too powerful in the hands of inexperienced programmers.

2.3 Analyzing the Post-change Situation

To analyze the situation after changes, we rely on observations made by the lecturer. During and after each lecture, the lecturer took notes reflecting on student attendance, engagement, questions, and effects of the new in-class exercises (Schon & DeSanctis, 1986). Additionally, we can consider feedback from two peer lecture shadowing sessions by colleagues from CS for lectures L4 and L5. Furthermore, we collected feedback from the students through a survey to be filled out by the end of each lecture. The survey was the prequel of a quiz with content-related questions for each lecture on the platform StudyNet used to distribute the materials. Students were asked to rate different aspects of the lecture on a scale of 1 to 5 (cf. Figure 3).

3 Results

3.1 The As-is Situation

The lecturer adopted the materials from its predecessor for teaching the course for the first time in 2022. The lectures correspond to L1–L5 from Table 1 (without the New In-class Exercises). There were only content-related online quizzes consisting of three to five multiple choice questions to be solved individually by the students at the end of a 90-minutes lecture.

The lecturer's experience confirms the hypothesized student disengagement: students were mostly passive and only consuming contents. Questions were rare and students were not engaging in discussions or interactions with the lecturer and peers. In total, 26 students were enrolled. The interest in the course dropped and active attendance decreased throughout the semester from approx. 13 to 7 students. The

rather one-sided lecturing parts led to a high workload for the teacher and the few students attending the lectures often left the impression of disinterest and fatigue.

The expert interviews with the previous lecturer and the lecturer of the course's second part confirmed the observations for previous instances of the course. The lecturers made similar observations and provided similar feedback regarding student engagement and interactions. The results from the expert lecture shadowing session for lecture L5 were as follows: the session was perceived as being rather monotonous with mostly the lecturer being active. Only a small number of students were paying attention but did not show much engagement. The lecturer had to depend on the same few students to get replies to questions.

3.2 The Post-change Situation

The lecturer revised and extended the existing materials with new in-class exercises according to the descriptions in Section 2.2 for the course in 2023.

In total, 16 students were enrolled in 2023. Compared to the course instance in 2022, student engagement generally increased in all dimensions (Trowler, 2010) due to the new in-class exercises as observed by the lecturer. Attendance stayed at a constant level of approx. 9 to 10 students for each of the five lectures. Most students worked actively on the in-class exercises, asked questions about them, engaged in discussions with their peers, the lecturer and the AI chatbot, and suggested solutions to the tasks. This increased engagement and more interactive, learning-oriented atmosphere also led to a workload decrease for the lecturer during a session.

Two colleagues from the CS faculty attended each a different session as peer shadowers. Both highlighted the interactive atmosphere during the in-class programming exercises as nice changes in between the lecturing parts. They appreciated the opportunity for students to directly apply the new lecture contents in the programming tasks to practice their understanding of the new materials with real-world datasets.

The average answer ratings of the *student survey* for questions Q1–Q8 related to each of the five lectures can be found in Figure 3. The bottom right part of Figure 3

provides a summary of the specific questions and rating scale. Note that due to the low number of students attending the classes and filling out the surveys (the ratings aggregate feedback from max. 7 students per lecture) the numbers should be rather interpreted as indications than as a solid statistical basis. The students see the new in-class exercises as valuable addition to the lectures to foster understanding and practical relevance of the new contents. The programming tasks balance the theoretical and practical lecture parts well and lead to an adequate degree of interactions with peers and the lecturer. In addition, we received informal written and oral feedback from students appreciating the in-class exercises fostering the understanding of the contents and helping with exam preparations.

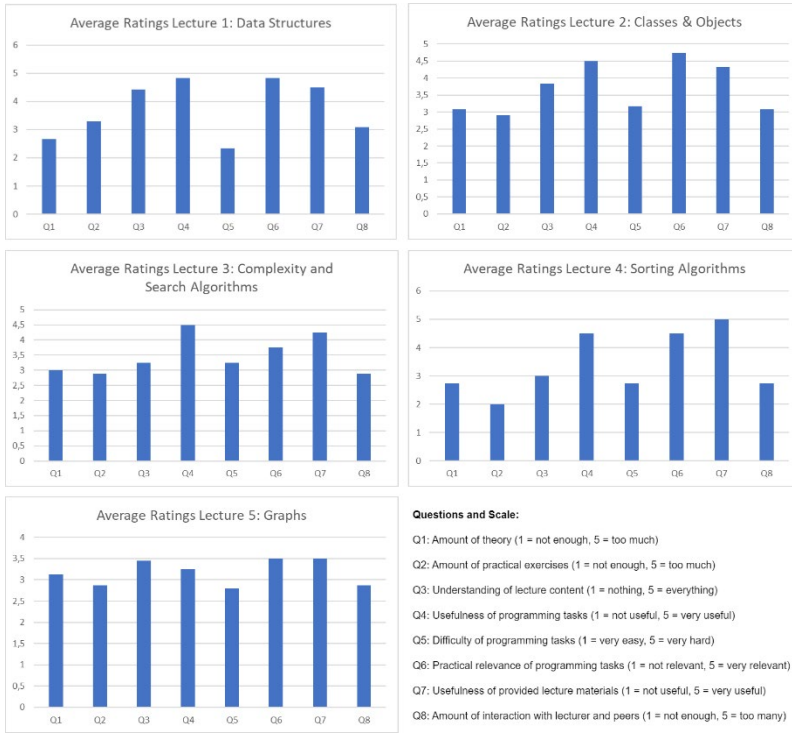

Figure 3. Results of study: average lecture ratings of questions Q1–Q8

4 Discussion

Analyzing the *as-is* situation, we see clear indications that the traditional front-of-class lecturing style does not foster student engagement (Barr & Tagg, 1995). Especially in small courses, the focus being on the lecturer presenting the contents may lead to a certain disengagement among the students who only play the passive role of consumers. Analyzing the *post-change* situation, we see clear indications of an improved learning atmosphere. The new interactive elements were well received as they foster understanding and application of the lecture contents in real world contexts using the smart factory datasets. From our observations and the user study, we can see that student engagement has increased due to the alternation of theoretical content and direct application in the new exercises. We can leverage the low numbers of students to create a more interactive learning atmosphere.

With these results we can answer the research question, which we introduced in Section 1. We find *short, content-related in-class programming exercises applied to real world use cases supported by AI* to be a suitable means to increase student engagement in fundamental computer science lectures for non-CS students. This requires students to be familiar with basic programming and provided with tools to solve the tasks. We find *Python* and *Jupyter* notebooks with task descriptions and, if necessary, already some code skeletons, good choices for non-CS students at the level of programming beginners. AI can be helpful in assisting novices with programming tasks, e.g., to look up and explain source code, but it should be used carefully when trying to generate larger code fragments. With these insights we can confirm the importance of *programming* as one of the most relevant means to learn and apply CS concepts (Borowczak & Burrows, 2019).

In the sense of SoTL, we reported the findings of studying our own teaching and students' learning with the goal of improving both in this project (Siegel, 2023; McKinney, 2010). While we were able to draw some conclusions from comparing the situation before and after the changes, we want to emphasize that this work is a SoTL *experience* report (Ludwig et al., 2018). The small group sizes of less than 11 attending students each year and the 1-year repetition cycle prevented us from conducting a comprehensive study. The results of this work can only be viewed as

indications, which limits their general applicability. Nevertheless, the results of the user study related to the post-change situation are promising as they indicate that the new in-class exercises nicely complement and balance the theoretical lecture contents with practical applications. We can hypothesize that the increased student engagement leads to a better understanding of the course contents and thus to a better student performance.

5 Conclusion & Future Work

In this work we investigated the design of programming tasks to increase student engagement in fundamental computer science lectures for non-CS students. This was motivated by the teaching of a course that was originally designed to be mostly taught front-of-class with almost no interactive elements. We followed the SoTL methodology to analyze the initial situation based on personal and external observations, which unveiled a low degree of student engagement. The results showed a clear need to intervene and move the students from passive content consumers to more active participants. We found short, content-related in-class programming exercises applied to real world use cases supported by AI to be a suitable means towards improving the learning atmosphere and increasing student engagement. Personal experience, external observations, and a small study with students confirmed the results. These insights may also benefit other CS-lecturers in similar situations who might be challenged with teaching the rather theory-heavy contents in a more interactive and engaging way.

The newly introduced in-class exercises can be regarded as a first step towards shifting the course from a classical teaching setting to a more modern learning format in future iterations following *Design Research* (Bakker, 2018). We will continue investigating which interactive formats and techniques could be suitable and applied to teach computer science fundamentals to non-CS students. All in all, a solid understanding of computer science concepts will also be beneficial for non-CS students when evaluating and deciding about the complexity and costs of adapting technologies related to AI and information systems.

References

Bakker, A. (2018). Design research in education: A practical guide for early career researchers. Routledge.

Barr, R. B., & Tagg, J. (1995). From teaching to learning—A new paradigm for undergraduate education. Change: The magazine of higher learning, 27(6), 12-26.

Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., & Santos, E. A. (2023, March). 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).

Bigatel, P., & Williams, V. (2015). Measuring student engagement in an online program. Online Journal of Distance Learning Administration, 18(2), 9.

Biggs, J., & Tang, C. (2010, February). Applying constructive alignment to outcomes-based teaching and learning. In Training material for "quality teaching for learning in higher education" workshop for master trainers, Ministry of Higher Education, Kuala Lumpur (Vol. 53, No. 9, pp. 23-25).

Borowczak, M., & Burrows, A. C. (2019). Ants go marching—Integrating computer science into teacher professional development with NetLogo. Education Sciences, 9(1), 66.

Brookfield, S. D. (2017). Becoming a critically reflective teacher. John Wiley & Sons.

Cerbin, W. (2018). Improving student learning from lectures. Scholarship of Teaching and Learning in Psychology, 4(3), 151.

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. Computer Science Education, 21(2), 135-173.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. Theory into practice, 41(4), 212-218.

Liebenberg, J., Huisman, M., & Mentz, E. (2015). The relevance of software development education for students. IEEE Transactions on Education, 58(4), 242-248.

Ludwig, H., Pilniok, A., Sethe, R., Szczyrba, B., & Vogel, M. (2018). Forschendes Lehren im eigenen Fach: Scholarship of Teaching and Learning in Beispielen (2. Blickpunkt Hochschuldidaktik), 125.

MacNeil, S., Kim, J., Leinonen, J., Denny, P., Bernstein, S., Becker, B. A., ... & Kumar, V. (2023, March). The Implications of Large Language Models for CS Teachers and Students. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education (Vol. 2).

Malburg, L., Seiger, R., Bergmann, R., & Weber, B. (2020, September). Using physical factory simulation models for business process management

research. In International Conference on Business Process Management (pp. 95-107). Cham: Springer International Publishing.

McKinney, K. (2010). Enhancing learning through the scholarship of teaching and learning: The challenges and joys of juggling. John Wiley & Sons.

Sanders, I., & Mueller, C. (2000). A fundamentals-based curriculum for first year computer science. ACM SIGCSE Bulletin, 32(1), 227-231.

Schon, D. A., & DeSanctis, V. (1986). The reflective practitioner: How professionals think in action.

Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-wesley professional.

Seiger, R., Malburg, L., Weber, B., & Bergmann, R. (2022). Integrating process management and event processing in smart factories: A systems architecture and use cases. Journal of Manufacturing systems, 63, 575-592.

Seiger, R. (2024, June 20). Python Jupyter Notebooks and Smart Factory Datasets for Interactive Exercises in Course Fundamentals of Computer Science. Zenodo. https://doi.org/10.5281/zenodo.12177764

Siegel, S. T. (2023). Scholarship of Teaching and Learning (SoTL): What else? Why (not)? How to? 2. Inverted Classroom and beyond 2023: Agile Didaktik fur nachhaltige Bildung, 35.

Sledgianowski, D., Gomaa, M., & Tan, C. (2017). Toward integration of Big Data, technology and information systems competencies into the accounting curriculum. Journal of Accounting Education, 38, 81-93.

Steiner, H. (2023). Hopscotch 4-SoTL: An Open-Access Tool for Generating a Well-Informed Research Design.

Trowler, V. (2010). Student engagement literature review. The higher education academy, 11(1), 1-15.

von Wangenheim, C. G., & da Silva, D. A. (2009). Survey on the relevance of topics in computer science education. Laboratorio Qualidade Produtividade Software, Univ. Vale Itajaı, Santa Catarina, Brazil, Rep. LQPS001. E, 9, 2009.