Recommendations to Create Programming Exercises to Overcome ChatGPT

1st Jonnathan Berrezueta-Guzman
Technical University of Munich
s.berrezueta@tum.de

2nd Stephan Krusche
Technical University of Munich
krusche@tum.de

***Abstract*—Large language models, such as ChatGPT, possess the potential to revolutionize educational practices across various domains. Nonetheless, the deployment of these models can inadvertently foster academic dishonesty due to their facile accessibility. In practical courses like programming, where hands-on experience is crucial for learning, relying solely on ChatGPT can hinder students' ability to engage with the exercises, consequently impeding the attainment of learning outcomes.**

This paper conducts an experimental analysis of GPT 3.5 and GPT 4, gauging their proficiencies and constraints in resolving a compendium of 22 programming exercises. We discern and categorize exercises based on ChatGPT's ability to furnish viable solutions, alongside those that remain unaddressed. Moreover, an evaluation of the malleability of the solutions proposed by ChatGPT is undertaken. Subsequently, we propound a series of recommendations aimed at curtailing undue dependence on ChatGPT, thereby fostering authentic competency development in programming. The efficaciousness of these recommendations is underpinned by their integration into the design and delivery of an examination as part of the corresponding course.

***Index Terms*—interactive learning, online training, education, assessment, plagiarism, autograder, large language models**

I. Introduction

ChatGPT gained popularity in the education field during the latter part of 2022 [1]. Its widespread use has been both lauded for its potential benefits and a subject of concern. ChatGPT provides access to information for solving various problems (mathematics, calculus, physics, etc.), and even generates written content by simply posing questions in natural language [2], [3]. It has also been employed for programming resolution across different programming languages [4].

ChatGPT can analyze a problem statement and provide a solution with an explanation of the provided code [5]. However, the utilization of this feature has sparked a contentious debate within the field of computer science education [6]. Concerns have been raised regarding the potential consequences of students relying solely on this tool without engaging in critical thinking or reflection when attempting programming exercises for homework or exams [7].

In this paper, we evaluate the efficacy of ChatGPT in solving programming exercises within an introductory programming course. The key contribution of this study lies in the formulation of recommendations that have been tested and proven effective for instructors in designing exercises within auto-assessment platforms. The applicability of these recommendations is demonstrated through their implementation in exercises designed for a final exam.

This paper is structured as follows: Section II provides a short explanation of Large Language Models (LLMs) and ChatGPT. Section III presents related work on ChatGPT in computer science education. Section IV describes the methodology employed in this study. Section V presents the results obtained. Section VI offers a comprehensive discussion along with recommendations based on the research findings. Finally, Section VII concludes the study and discusses potential avenues for future research.

II. Large Language Models

Large Language Models (LLMs), employing unsupervised learning on voluminous textual datasets, have revolutionized Natural Language Processing (NLP) tasks such as suggesting feedback in textual exercises in large courses [8], generation of feedback on textual student answers [9], automation of grading textual student submissions [10] and text generation, though they present ethical quandaries including biases and misinformation [11], [12].

Utilizing transformer architectures, LLMs such as OpenAI's Generative Pre-trained Transformer (GPT) series are adept at handling long-range dependencies in language modeling due to their attention mechanisms [13], [14].

OpenAI's GPT-4 possess significantly augmented capabilities for handling complex NLP tasks [15], [16]. The integration of reinforcement learning, utilizing reward signals for decision optimization through agent-environment interactions, further bolsters the applicability of LLMs in recommendation systems [17], [18].

The progression of LLMs underscores the escalating relevance of NLP in AI, striving for sophisticated human-machine interactions [19].

III. Related Work

ChatGPT has sparked controversy within the education field. Jalil and his colleagues evaluated ChatGPT's performance in answering common questions from a popular software testing curriculum [20]. Results indicate that ChatGPT provides correct or partially correct answers in 55.6 % of cases, achieving 53.0 % accuracy in providing explanations. When prompted in a shared question context, the tool exhibits a slightly higher rate of correct responses. These findings suggest potential benefits, such as ChatGPT guiding students through exercises to enhance their comprehension. Notably, ChatGPT performs best with coding questions, achieving 83.3 % accuracy, followed by conceptual questions at 55.6 %, and combined questions at 31.3 %.

In another study [21], researchers analyzed response quality in ChatGPT. While responses were often reliable, there were instances with misleading information. The study concluded that while the output quality of ChatGPT is acceptable, there is room for improvement. Notably, when generating code from problem statements, the generated code failed to function correctly when compiled. Participants in the experiment reported that ChatGPT's answers were somewhat accurate but not entirely helpful in solving specific coding problems. They expressed the need to rely on personal experience and trial and error. One participant suggested that more specific specifications would lead to more accurate code generation.

Lau and Guo analyzed cheat detection in a CS1 course using an autograder [22]. They found that ChatGPT can easily generate programming code from English specifications. Results showed that ChatGPT completed approximately 45 hours of work in just about 1 hour through mostly mindless copy-pasting, achieving an average score of 96 % based on the auto-grading system. However, the study also observed that the generated code style often deviated from the class and book style, utilizing untaught constructs like `while(true)` loops with breaks. The instructors were unable to train ChatGPT to adhere to their class's specific coding style.

Furthermore, when a single student utilizes ChatGPT across multiple programs, the code style may differ in ways that regular students do not exhibit, such as varying indentations or line spacing. Additionally, although ChatGPT generates different solutions for different students, it may start to produce solutions with minor variations, such as variable name changes. Consequently, students may face accusations of similarity or plagiarism as ChatGPT cannot generate an infinite number of distinct solutions.

IV. Methodology

In this study, we evaluated ChatGPT's performance (version 3.5 and 4) in solving programming exercises within Artemis, an automatic grading platform for interactive learning [23], [24]. The analysis considered the average time invested and grades achieved by students and the instructor (using ChatGPT). We identified key factors that determine the usefulness or limitations of ChatGPT in programming assignments.

A. Course Selection

We analyzed 22 exercises from an introductory programming course offered in the first semester of the Information Engineering bachelor's degree program at the Technical University of Munich (TUM). 65 students participated in the course. It lasted 12 weeks in the winter semester 2022-2023. We focused specifically on the homework exercises, which students had to solve independently and which accounted for the practical component of the course grade. These exercises were evaluated using Artemis' integrated plagiarism check tool to detect any instances of academic misconduct [25].

The theoretical assessment of the course is a final exam, where 80 % of the grade is based on solving programming exercises, and the remaining 20 % is allocated to quiz exercises. The examination follows the recommendations outlined in this paper. The outcomes are analyzed and discussed.

B. ChatGPT Evaluation

The instructor evaluated the 22 homework exercises using ChatGPT. The process involved documenting the time spent on copying and pasting the problem statement, the number of passed tests, and the points earned. If ChatGPT failed to achieve a perfect grade in a single attempt, the time required to complete the exercise by adapting the provided code was recorded. Additionally, the reasons behind ChatGPT's inability to complete the exercise were noted. This data will be used to compare the performance of students with the use of ChatGPT.

V. Results Analysis

The grades obtained by the students in each assignment are averaged and compared with the grade obtained by the instructor using ChatGPT in a single attempt (without adapting the obtained code) and are compared in Figure 1.

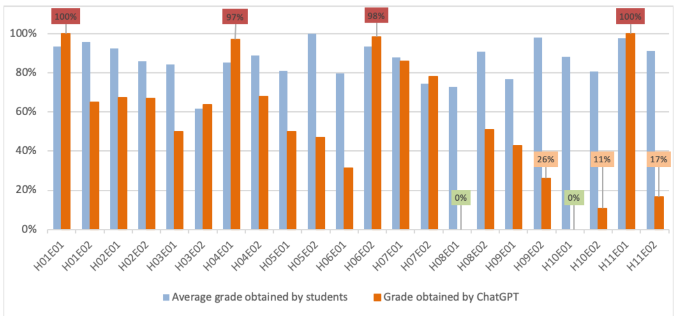

Fig. 1. Comparison between the average grade of students in each exercise with the grade obtained by ChatGPT in one go (without code adaptation). Green label: exercises that are impossible to solve by Chat GPT; Orange label: Exercises very difficult to solve by Chat GPT; Red label: Exercises that ChatGPT can solve in one go at 100 %.

Based on the results shown in Figure 1, we could answer which exercise are easy, difficult and impossible for ChatGPT to solve.

A. What is too easy for ChatGPT?

We identified 4 exercises in that ChatGPT obtained a better grade than the average grade obtained by the students.

H01E01 - Space Competition. This is a basic exercise for students to become familiar with the Integrated Development Environment (IDE) where students are required to output some text. Therefore, ChatGPT is expected to be able to solve it completely.

H04E01 - Panic at Burger House. This exercise is based on the use of the Java collection types, Lists, Stack, and Queue. It provides a UML diagram to help students recognize the program structure. From Artemis, it is easy to copy this diagram and paste it into the ChatGPT input area because takes the code behind it (SVG image). ChatGPT got everything to provide a solution that gets 100 % of the grade.

H06E02 - TUM Supermarket. Students must implement generic data structures. Like H04E01, the given UML diagrams make it easy for ChatGPT to create the solution structure code. In addition, ChatGPT can solve the tasks separately.

H011E01 - Penguins and Recursions. The problem statement is very formalized and does not require interpretation and logical implementation. Therefore, ChatGPT solves it perfectly. The only challenge for a student is to copy Artemis formulas to the ChatGPT input area.

B. What is very difficult for ChatGPT?

We identify that 3 exercises did not obtain more than 30 % of the maximum score.

H09E02 - JSON Jobs. ChatGPT understood the exercise wrongly. It ignored the hints and used other methods. Additionally, the implementation of a new library forces the student to adapt to the solution provided by ChatGPT. The required time to adapt the exercise represents almost the same time to solve it without ChatGPT.

H010E02 - Calculator. This exercise is based on the creation of a basic calculator. The GUI part is specified by images (PNG files). The template provides some classes and attributes that should be used to develop the graphical part and the logic part (calculation methods). Once again, ChatGPT provides arbitrary classes and methods. The student would spend a lot of time adapting the code.

H011E02 - TUM Triangle. It involves the integration of GUI and recursion. The specification is conveyed by a GIF file that illustrates the program's behavior rendering ChatGPT incapable of solving this exercise.

C. What is impossible for ChatGPT?

We found that 2 exercises cannot be solved by ChatGPT.

H08E01 - Extending the Game. This exercise builds upon a preceding one (H03E02 - Fundamentals of a game) with a pre-existing solution provided in the template. The intricate nature of the UML diagram perplexes ChatGPT due to its lack of context. Consequently, the code generated by ChatGPT is arbitrary and markedly distinct from the template's code. Students must refer to the template and review the given implementation to successfully fulfill the exercise.

H10E01 - GUI Upgrade for the Game. Similar to H08E01, this exercise relies on the solution of a prior exercise. The template already contains the solution, and the student's task is to implement the GUI component. However, ChatGPT lacks awareness of this requirement, leading to confusion and the provision of an implementation that fails. This occurs because ChatGPT employs arbitrary parameters and methods.

D. Learning outcome concerns

By analyzing the grades in Figure 1 we obtain that the overall average grade for the course is 86.4 %, while the overall average grade obtained by the instructor using ChatGPT in one-go is 55 % (minimum grade for passing is 50 %). We can deduce that a student using ChatGPT could pass an introductory programming course without any reinforcement at all. This would lead to problems later on in subsequent subjects in the Bachelor's program.

VI. Findings

The learning of programming depends mostly on the practice [26] and ChatGPT seems to represent a threat to this.

Finding 1: With no change in usage, ChatGPT facilitated a 91 % reduction in time invested in solving 22 assignments. Refining the code for full grading still resulted in time savings of 74 % below the student average.

This finding poses a significant pedagogical challenge, as programming proficiencies are inherently iterative and necessitate substantial practice for maturation [27]. Therefore, based on the analysis of the results, we provide a set of recommendations o create programming exercises to overcome ChatGPT.

A. Recomendations

1) Reduce the problem statement. It should be concise, avoiding excessive details and excluding project structure explanations. Consequently, ChatGPT lacks sufficient context and may generate arbitrary code based on its understanding. As a result, students must modify the provided code to align with their requirements.

2) Add TODOs comments in the template. Distributing them across multiple files (classes) complicates the process for students to gather all the required information in a single location for direct copy-pasting into ChatGPT.

3) Provide a preview implementation. Template files include partial implementation. This approach compels students to review not only the problem statement but also the template, familiarizing themselves with the programming style and correct program structure. Notably, exercises like H08E01 and H10E01 prompted students to question whether it was more advantageous to solve them using ChatGPT or independently. Our analysis revealed that adapting the arbitrary code generated by ChatGPT in these exercises required more time than solving them manually.

4) Use image files in the problem statement. PNG, GIF, or JPG files as visual representations of the resulting program make ChatGPT unable to interpret the context, resulting in the generation of arbitrary code. As a consequence, students may question the usefulness of ChatGPT when it comes to adapting the code based on these visual representations.

5) Use hidden test cases. Test cases are revealed to students only after the deadline. This practice compels students to independently verify the correctness of their solutions and not solely rely on continuous feedback.

6) Use dynamic test cases. where the input data varies with each test execution. This approach requires students to develop robust solutions that can handle diverse conditions, instead of relying on a static solution generated by an AI. However, it may involve additional effort for the instructors.

7) Evaluate efficiency. Monitor the runtime and memory usage of students' programs to emphasize the importance of efficiency. This approach ensures that submissions not only meet the functional requirements.

8) Assess the quality of the code. Such as adherence to programming style guidelines, understandability, and modularity. Students will write their code carefully and reflectively, rather than simply copying an AI-generated solution.

9) Ask for documentation. Require students to provide code comments and written documentation outlining their solution. This practice fosters comprehension of the assignment and discourages direct copying of AI-generated solutions.

10) Customize the task. Incorporate explicit exercise requirements that diminish the likelihood of a typical AI-generated solution. This can be achieved by stipulating the utilization of specific algorithms or data structures.

11) Do a plagiarism check. This approach aids in identifying potential AI-generated or plagiarized solutions. It is important to note that while ChatGPT can generate multiple solutions, they may exhibit similarities or repetitions after a certain number of iterations.

Depending on the exercise, these recommendations can be applied to mitigate the use of ChatGPT by students. Therefore, we have created a final exam to test these recommendations.

B. Application of the recommendations in the preparation of individual examination

The final examination incorporates these recommendations. The exam consists of a single quiz exercise and three programming exercises, accounting for 20 % and 80 % of the total evaluation weightage, respectively.

TABLE I
INDIVIDUAL EXAMINATION EXERCISES.

Exercise	Grade percentage	Type
Quiz exercise	20 %	Questions
Object-Oriented Programming	18 %	Coding
Graphical User Interface	35 %	Coding
Streams	27 %	Coding

1) Object-Oriented Programming. This exercise follows recommendations 1-5, and 11. The problem statement and the principal task of this exercise are reduced. The provided UML diagram is a PNG file. Additionally, the provided template implements already some classes (3 out of 8) of the UML diagram. Finally, a plagiarism check is conducted.

2) Graphical User Interface. This exercise also follows the previous recommendations. The problem statement is reduced and 2 graphics are presented: registration form view and information view. Additionally, one animated GIF is presented where students can see the behavior of the controls (buttons, text fields, and labels). The template provides the attribute's names and the structure. The student must complete the implementation following the *TODO* comments in the template code.

3) Streams. This exercise incorporates prior recommendations by reducing the problem statement and including a UML diagram as a PNG file. The provided template already includes partial implementation, prompting students to compare the UML diagram with the code and complete the implementation according to the provided *TODO* comments.

Finding 2: Exercises that followed these recommendations were identified as unsolvable with ChatGPT alone because the time required for code adaptation significantly exceeded the expected solution time without ChatGPT (approximately 30 % more).

The exam did not present real-time feedback; it solely indicates whether students' code successfully compiles or not. Additionally, it was conducted onsite under supervision to prevent any form of academic dishonesty. The average grade is shown in Table II.

TABLE II
EXAMINATION RESULTS.

Exercise	Grade average percentage
Quiz exercise	68.8 % (13.4 / 20)
Object-Oriented Programming	64.9 % (11.7 / 18)
Graphical User Interface	53 % (18.6 / 35)
Streams	52.8 % (14.3 / 27)

Finding 3: Grade and pass rate disparities between practical (1.7, 73.8 %) and final exam (2.7, 54.4 %) components, coupled with negative plagiarism checks, indicate ChatGPT exploitation in practical exercises.

However, conclusively proving this hypothesis poses significant challenges and the methodology presented in [22] would require significant instructor effort. Future work involves developing a feature in Artemis that can detect ChatGPT usage or provide guidelines to create exercises that are resistant to ChatGPT. Meanwhile, the following additional recommendations can be considered:

1) Plagiarism detection software can help to identify similarities and detect cheating. However, it is not able to detect if students use ChatGPT for coding [28], [29].

2) Manual code review allows to detect unusual patterns, style inconsistencies, or suspicious comments. This method can be time consuming and is only effective in smaller groups.

3) Comprehension questions and code reviews can ensure that students understand the submitted code and its concepts.

4) Assignment variation makes cheating more difficult. However, this could work only with a small group of students.

5) Pair or group work encourages student engagement and facilitates knowledge sharing among peers. However, it is crucial to ensure that each student actively contributes and avoids undue dependency.

6) Monitoring online resources can identify suspicious activities, such as sharing of homework solutions on platforms, forums, and social media, but might be time consuming.

VII. Conclusion and Future Work

This paper analyzed how well GPT 3.5 and 4 can solve several typical student tasks in a first semester programming course. Based on this analysis, it categorizes tasks that are easy, difficult, and impossible for ChatGPT to solve. It derives recommendations and best practices for instructors to verify academic integrity and minimize opportunities for deception.

Employing the provided recommendations in the creation of exercises for an exam (that included images, concise problem statements, and template code) hindered students from using ChatGPT, as they couldn't simply copy and paste the requirements. These recommendations can be applied to other programming courses and exercises involving text or model generation, such as UML diagrams.

We perceive ChatGPT as a tool that can negatively affect learning outcomes in programming courses if misused. Students who solve exercises without analyzing and reflecting on the problem may face long-term issues in their academic and professional lives. Therefore, this paper contributes with suitable recommendations to prevent self-deception resulting from improper ChatGPT use.

While some universities have banned ChatGPT due to concerns over cheating, others have adopted approaches with clear guidelines. Given the evolving nature of AI, further discussion and research on ethical implications are necessary. Universities need to develop policies for the responsible and ethical use of AI models like ChatGPT, clearly communicating the ethical aspects and setting defined boundaries [30].

We think that AI tools should be used to complement personal effort and learning, not replace it. Establishing principles, policies, and codes of conduct is essential to ensure that the role of AI in improving learning and support is both responsible and ethical.

We consider that future LLMs (e.g., GPT-5, or fine-tuned versions specific for education) may become even more powerful. Then, the recommendations provided in this paper need to be re-evaluated and adapted accordingly. In the future, we plan to analyze how valuable LLMs can support the learning experience in project-based software engineering courses with more creative programming tasks, for example based on chaordic learning [31].

References

[1] E. Kasneci *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, 2023.

[2] B. D. Lund and T. Wang, "Chatting about chatgpt: how may ai and gpt impact academia and libraries?," *Library Hi Tech News*, vol. 40, no. 3, pp. 26–29, 2023.

[3] L. Bishop, "A computer wrote this paper: What chatgpt means for education, research, and writing," *Research and Writing*, 2023.

[4] N. M. S. Surameery and M. Y. Shakor, "Use chat gpt to solve programming bugs," *International Journal of Information Technology & Computer Engineering*, vol. 3, no. 01, pp. 17–22, 2023.

[5] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," *arXiv preprint:2301.08653*, 2023.

[6] S. Biswas, "Role of chatgpt in computer programming.: Chatgpt in computer programming.," *Mesopotamian Journal of Computer Science*, vol. 2023, pp. 8–16, 2023.

[7] D. R. Cotton, P. A. Cotton, and J. R. Shipway, "Chatting and cheating: Ensuring academic integrity in the era of chatgpt," *EdArXiv*, 2023.

[8] J. P. Bernius, S. Krusche, and B. Bruegge, "A machine learning approach for suggesting feedback in textual exercises in large courses," in *8th Conference on Learning at Scale*, pp. 173–182, 2021.

[9] J. P. Bernius, S. Krusche, and B. Bruegge, "Machine learning based feedback on textual student answers in large courses," *Computers and Education: Artificial Intelligence*, vol. 3, 2022.

[10] J. P. Bernius, A. Kovaleva, S. Krusche, and B. Bruegge, "Towards the automation of grading textual student submissions to open-ended questions," in *4th European Conference on Software Engineering Education*, pp. 61–70, 2020.

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint:1810.04805*, 2018.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[16] O. Analytica, "Gpt-4 underlines mismatch on ai policy and innovation," *Emerald Expert Briefings*, 2023.

[17] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] A. Koubaa, "Gpt-4 vs. gpt-3.5: A concise showdown," 2023.

[19] M. Binz and E. Schulz, "Using cognitive psychology to understand gpt-3," *National Academy of Sciences*, vol. 120, no. 6, 2023.

[20] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, and W. Lam, "Chatgpt and software testing education: Promises & perils," in *International Conference on Software Testing, Verification and Validation Workshops*, pp. 4130–4137, IEEE, 2023.

[21] A. Tlili, B. Shehata, M. A. Adarkwah, A. Bozkurt, D. T. Hickey, R. Huang, and B. Agyemang, "What if the devil is my guardian angel: Chatgpt as a case study of using chatbots in education," *Smart Learning Environments*, vol. 10, no. 1, p. 15, 2023.

[22] S. Lau and P. Guo, "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," 2023.

[23] S. Krusche and A. Seitz, "ArTEMiS: An Automatic Assessment Management System for Interactive Learning," in *49th Technical Symposium on Computer Science Education*, pp. 284–289, ACM, 2018.

[24] S. Krusche and A. Seitz, "Increasing the interactivity in software engineering moocs - A case study," in *52nd Hawaii International Conference on System Sciences*, pp. 1–10, 2019.

[25] S. Krusche, "Interactive learning - A Scalable and Adaptive Learning Approach for Large Courses," Habilitation, Technical University of Munich, 2021.

[26] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer science education*, vol. 13, no. 2, pp. 137–172, 2003.

[27] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão, "A systematic literature review on teaching and learning introductory programming in higher education," *Transactions on Education*, vol. 62, no. 2, pp. 77–90, 2018.

[28] L. Prechelt, G. Malpohl, M. Philippsen, *et al.*, "Finding plagiarisms among a set of programs with jplag.," vol. 8, no. 11, p. 1016, 2002.

[29] D. Rusch, T. Lancaster, and A. Gervais, "Detecting source code plagiarism from online software repositories," 2022.

[30] D. Mhlanga, "Open ai in education, the responsible and ethical use of chatgpt towards lifelong learning," 2023.

[31] S. Krusche, B. Bruegge, I. Camilleri, K. Krinkin, A. Seitz, and C. Wöbker, "Chaordic Learning: A Case Study," in *39th International Conference on Software Engineering: Software Engineering Education and Training*, pp. 87–96, IEEE, 2017.