BOARD # 90: WIP: Evaluating Programming Skills in the Age of LLMs: A Hybrid Approach to Student Assessment

Mr. Joshua Coriell, Louisiana Tech University
Ankunda Kiremire, Louisiana Tech University

Dr. Ankunda Kiremire is a senior lecturer of Computer Science and Cyber Engineering at Louisiana Tech University and serves as the program chair for its Computer Science Department. His research interests include Computer Science Education, Cyber Security, and Data Science.

Dr. Krystal Corbett Cruse, Louisiana Tech University

Dr. Krystal Corbett is the First-Year Engineering Programs Coordinator and Assistant Professor in the Mechanical Engineering Department at Louisiana Tech University. She is also the Director of the Office for Women in Science and Engineering at Louisiana Tech.

William C. Long, Louisiana Tech University

WIP: Evaluating Programming Skills in the Age of LLMs: A Hybrid Approach to Student Assessment

Abstract

The advent of large language models (LLMs), such as OpenAI's ChatGPT, has augmented the challenge of assessing student understanding and ensuring academic integrity is maintained on homework assignments. In a course with a heavy focus on programming, it is common to have a significant portion of the grade be determined by such assignments. When an LLM is prompted with the instructions for a programming assignment, it can readily give a solution that required little to no thought from the student. This has made accurately assessing a student's programming skills through homework assignments significantly more challenging.

This work-in-progress paper investigates the student experience of the transition from solely at-home programming assignments to at-home programming assignments with the addition of three handwritten components in the form of in-class programming assessments. A key piece of this transition is being cautious to not add additional work to the professor's workload by requiring additional assignments. To offset the addition of assignments, the at-home assignments were converted to automatically graded assignments using Gradescope. These changes were implemented in a pilot session in the 2024-2025 academic year. In the transition, the total programming grade remained at 20% of the course grade, however, three-fourths of this percentage is now determined by the in-class assessments, reducing the portion of the course grade that could potentially be determined through the students' use of LLMs from 20% to 5%.

To assess the effects of this change on student experiences, the students enrolled in the pilot course were surveyed on the clarity, difficulty, and effectiveness of the assignments, as well as the accuracy, fairness, and timeliness of the autograder. Instructors who taught the course were interviewed to assess the professor experience with the transition. The responses from the surveys, interviews, and student performance, provide a baseline for future adjustments to the three-course sequence to accurately assess students on their basic programming skills in a world where LLMs are becoming more prevalent.

Introduction

In recent years, large language models (LLMs) have rapidly developed [1] and have quickly become a daily resource for professionals and students alike [2]. LLMs are able to enhance the learning experience through rapid content generation [3]. One study that surveyed students to better understand their use of LLMs revealed that some utilized them to answer questions on homework and exams, write outlines and essays, and identify errors in code. They also identified what they felt were ethical uses of LLMs, which included helping to understand concepts, correcting grammar, and creating citations, among others. When pressed, students revealed stress, running out of time, and failing to find the answer for themselves pushed them to using LLMs in ways that may seem unethical [4].

In a computer science course, LLMs can be used to both generate code and help a student understand it [5]. Depending on how the LLM is being leveraged, it could be perceived as a benefit or risk to the student [6]. During their first year, many computer science students learn the fundamentals of programming, which serves as a critical foundation for their future computer science courses. However, as they encounter difficult programming challenges on a homework assignment, students may be tempted to use LLMs to quickly generate a solution in order to turn the assignment in and get a grade. This approach can have a negative effect on the student, which could be exacerbated as they progress through their computer science curriculum. As students rely on LLMs to generate full code, they may not be learning the content effectively [6]. Additionally, from the instructor's perspective, distinguishing whether a student's submitted code was generated by an LLM or written by a human can be challenging. Furthermore, the reliability of the tools and programs designed to detect such cases remains uncertain. In courses where at-home programming assignments are a large portion of a student's course grade, this leaves the possibility of poor assessment of a student's skill as a programmer.

To better assess students in a world with easily accessible LLMs, instructors are looking to find alternative ways to deliver and grade programming assessments. One such solution is to reduce the weight of at-home programming assignments. Another is to only do in-class programming assessments, whether it is on a locked-down device or on paper. In this work in progress, we investigate the student and faculty perspectives at Louisiana Tech University on the implementation of a hybrid of these two approaches in the 2024-2025 school year. More specifically, the approach involved a reduction of the percentage of the grade that is determined through at-home programming assignments along with the introduction of paper-based programming assessments. Overall, the percentage of the grade determined by programming assignments did not change.

Course Background and Modifications

At Louisiana Tech University, first-year computer science and cyber engineering students participate in a three-course series that emphasizes programming fundamentals, computer architecture, algorithms, and data structures. To evaluate their programming skills, each course includes several at-home programming assignments that collectively account for 20% of the final grade. Alongside these assignments, students take three paper-based, in-class exams per course, which make up 50% of the overall grade. These exams are designed to assess the material covered in the units leading up to the exam and are scheduled to be completed within a single class period.

While the courses heavily focus on programming, they also cover broader topics that may not directly involve coding. For instance, an exam might omit programming content if the preceding material focused on theoretical or conceptual topics, such as Introduction to Computer Architecture. Despite this, students are expected to continuously practice and improve their programming skills throughout the term. To this end, they are assigned frequent programming tasks of increasing difficulty, ensuring steady progression.

The grading structure for these courses comprises the aforementioned exams and programming assignments, as well as additional components such as puzzles designed to sharpen problem-solving skills and larger programming projects. These projects not only challenge students technically but also foster collaboration and the development of soft skills critical for their future careers.

This study was designed with minimal disruption to existing courses in mind. Our primary goal was to ensure that the number of assignments and their weight toward a student's grade remained largely unchanged. This consistency allowed for uniform grading and facilitated a straightforward comparison between sections implementing the modified structure and those following the original format.

A secondary goal was to minimize the additional workload for instructors adopting the revised course structure. Instructor workload was measured by the time required to grade programming assignments over a 10-week term. In the traditional structure, instructors typically graded 15–20 assignments per term for each of the 30–40 students in a single class section. Maintaining a manageable grading workload was, therefore, a critical consideration in the design of our modifications.

To achieve these goals, we introduced two key adjustments to the course structure:

1. The addition of paper-based programming quizzes.
2. The conversion of existing programming assignments into automatically graded assignments

Paper-based programming quizzes

Three paper-based programming quizzes were introduced to evaluate students' understanding of the programming concepts covered in their assignments. Each quiz was designed to be completed in under 10 minutes and focused on the material students had encountered in the programming assignments up to that point in the term.

The number of quizzes was chosen to align with the number of exams in the original course structure. By pairing a quiz with each exam, we were able to administer both during the same class period, ensuring that the original schedule remained unchanged.

To integrate the quizzes into the grading system, we redistributed the weight of the programming component from the original structure. Previously, programming assignments accounted for 20% of the total grade. In the revised structure, programming quizzes comprised 15% of the grade, with each quiz contributing 5%.

Autograded programming assignments

To address the increased workload created by the programming quizzes, we adapted the existing programming assignments to be automatically graded using Gradescope [7], web-based software that integrates with our institution's Learning Management System (LMS).

The use of automatically graded assignments offered two key benefits:

1. Improved Feedback Loop: Students were able to submit multiple versions of their assignments and receive frequent, tailored feedback at any time while the assignment was open.
2. Reduced Grading Effort: By eliminating the need for instructors to set up and test each student's submission, the time spent on grading was significantly reduced. This allowed instructors to focus on providing more personalized feedback on programming quizzes and other aspects of the course.

In the revised structure, the autograded programming assignments accounted for 5% of the overall grade. Combined with the 15% allocated to the programming quizzes, the original 20% contribution from programming assignments remained unchanged.

While the autograded assignments were still potentially vulnerable to cheating, including the use of large language models (LLMs), this risk was mitigated by their limited grade weight (5%) and the requirement for students to create programmatic solutions by hand during the programming quizzes. This ensured that students spent more time engaging with the underlying concepts the programming assignments were designed to teach rather than relying solely on external tools or shortcuts.

Data Collection

The primary goal of this paper is to present both student and instructor perspectives on the implementation of the programming assignment assessment approach. As this is a work-in-progress, data collection has only been conducted during one term at the time of writing. The group surveyed consists of 26 honors students enrolled in the first course of a three-course sequence. The students are primarily in Computer Science and Cyber Engineering, with two students from other disciplines.

To gather student perspectives, surveys were administered at two points during the term: once after the first programming assignment and again after the final assignment. The first survey asked general questions about the assignment, including its difficulty, the time required, and its appropriateness. Additionally, it contained questions focused on the autograder tool, assessing students' views on its ease of use, accuracy, and the quality of feedback it provided during the assignment. The survey also included questions about the quality of the instructions included in the programming assignment on the use of the autograder.

The final survey, administered at the end of the term, included similar questions, but they were rephrased to capture students' opinions on all programming assignments throughout the course. This survey also asked students to identify the most and least difficult assignments, as well as which assignments were easiest or hardest to use with the autograder.

Both surveys included a combination of Likert scale and open-ended questions.

Preliminary Results and Discussion

All students in the section participated in the post-assignment survey. Of these, 92% felt that the autograded assignments were appropriately aligned with the material covered in class and the concepts they were expected to master by that point in the term. However, only 69% shared the same view regarding the paper programming quiz, while 29% were neutral about its appropriateness. Although both results were largely positive, this discrepancy highlights a gap in content coverage between the autograded assignments and the programming quizzes, which will be addressed in future classes.

When asked about the scheduling of the programming quiz and exams, 76% of students preferred having both during the same class period, rather than separating them into different days. As one student put it, "I'm already in test-taking mode and ready for it. It would just add extra stress to have two separate days of testing."

Regarding the difficulty of adjusting their code for compatibility with the Gradescope autograding suite, 73% of students did not find it challenging, while 19% did. The remaining students were neutral in their responses.

However, only 42% of students found the autograder feedback helpful, indicating that more attention should be given to designing programming tests that accommodate the variety of student submissions.

Despite this, 80% of students felt that the autograder graded their submissions accurately.

Overall, most students expressed appreciation for the ability to use the autograder for their assignments and valued the quick feedback, which allowed them to see their grades promptly.

Future Work

This study provided preliminary insights into the student perspectives of the programming assignments and the assessment approach for a first-year computer science course. Further analysis could provide a more concrete understanding of the impact of the autograder tool on the students' overall course experience. Looking at each programming assignment in depth and evaluating the student experience could yield more insights. Analyzing the survey results further through statistical analysis can provide valuable insights into the impacts of the various changes to the course discussed in this study. Additionally, looking at the students' performance in future classes where the fundamental content from this course is a prerequisite could illustrate the benefits and impacts of the autograder. This study will continue through additional phases of implementation in subsequent courses as well as expanding to include multiple instructors.

References

[1] Wang, Z., Chu, Z., Doan, T.V. *et al.* History, development, and principles of large language models: an introductory survey. *AI Ethics* (2024). https://doi.org/10.1007/s43681-024-00583-7

[2] Saiedian, H. (2024, July), *Leveraging Large Language Models in Education: Enhancing Learning and Teaching* Paper presented at 2023 ASEE Midwest Section Conference, University of Nebraska-Lincoln, Lincoln, Nebraska. 10.18260/1-2-119-46353

[3] Haikal, T., & Lightfoot, R. H. (2024, March), *Enhancing Education Through Thoughtful Integration of Large Language Models in Assigned Work* Paper presented at 2024 ASEE-GSW, Canyon, Texas. 10.18260/1-2--45377

[4] Paustian T and Slinger B (2024) Students are using large language models and AI detectors can often detect their use. Front. Educ. 9:1374889. doi: 10.3389/feduc.2024.1374889

[5] F. A. Pirzado, A. Ahmed, R. A. Mendoza-Urdiales and H. Terashima-Marin, "Navigating the Pitfalls: Analyzing the Behavior of LLMs as a Coding Assistant for Computer Science Students—A Systematic Review of the Literature," in *IEEE Access*, vol. 12, pp. 112605-112625, 2024, doi: 10.1109/ACCESS.2024.3443621.

[6] Menekse, M. (2023), Envisioning the future of learning and teaching engineering in the artificial intelligence era: Opportunities and challenges. J Eng Educ, 112: 578-582. https://doi.org/10.1002/jee.20539

[7] J. Tharmaseelan, K. Manathunga, S. Reyal, D. Kasthurirathna and T. Thurairasa, "Revisit of automated marking techniques for programming assignments", IEEE Global Engineering Education Conference EDUCON, vol. 2021-April, 2021