Revealing Student Understanding By Using ChatGPT in Computer Science Assignments

Lorraine Jacques, University of Tampa

Abstract: In recent months, there has been much concern about students using ChatGPT and similar tools to do their assignments, but what if we as instructors incorporate these tools deliberately? What can the resulting student work tell us about their understanding and misconceptions? Two programming assignments using ChatGPT were given to students in a CS 101 course; the resulting student work revealed insights into student learning in ways that are difficult to capture in traditional programming assignments. A qualitative analysis of this data suggests the following about student learning: (1) Students prefer to represent code using a flowchart but have more errors in their representation than those who choose pseudocode, (2) Students are less likely to error-check the input when modifying existing code than they when incorporating a pre-generated subroutine into a program they started themselves, and (3) Creating a different approach to an AI-generated solution reveals misconceptions student have concerning topics covered much earlier in the course.

One day in fall, 2022, a student came to my office to show me this new tool, ChatGPT. More specifically, to show me that ChatGPT could create very good code for half of the programming assignments in our introductory computer science course. Obviously, my initial thought was, "What do we do now?," a thought that was echoed throughout computer science departments everywhere. Banning these tools would not only be futile, it would be a disservice to our students since over 80% of software professionals are using generative AI to write code (Kolakowski, 2023). How to use them in the CS classroom, however, has instructors divided. Many have not used generative-AI themselves (Prather et al., 2023) and most worry that students will blindly generate code without taking time to understand it (Prather et al., 2023; Zastudil et al., 2023). But to successfully program using generative AI, students need instruction not only in the core CS concepts but also how to choose between various solutions to identify the "best fit" for their problem (Becker et al., 2023).

As a former math educator, I realized that mathematics has already survived a similar technological revolution with the graphing calculator, so I developed two programming assignments that apply current math pedagogy to computer science education (Jacques, 2023). Unexpectedly, when I used these assignments in my introductory computer science course, I learned more about what my students understood or still misunderstood than I have with more traditional assignments. This paper will share the results of qualitatively analyzing the student work from these assignments.

Math pedagogy

Convergent Cognition Theory suggests that new knowledge in one domain can be built from prior knowledge in another domain and vice-versa (Rich, Bly, & Leatham, 2014). This reciprocal effect happens because both domains share core attributes, but learners find that one is more abstract and the other is more applied. Mathematics and computer science share this relationship because, "Computer science inherently draws on mathematical thinking" (Wing, 2006). In most of the literature, this theory is realized through the application of computer programming to learn mathematics (e.g., Aydin, 2005; Harel & Papert, 1990; Kafai, 1996). Being a reciprocal relationship, however, suggests that one could apply math pedagogy to computer science education.

Having students create alternate representations of an algorithm, explain someone else's solution, or develop a different approach to solving a problem are strategies for developing critical thinking skills in math education research (Khalid et al., 2020; Laursen & Rasmussen, 2019; NCTM, 2014; NRC, 2001). Applying these strategies to computer science education has the same benefits (Blackwell et al., 2001; Corney et al., 2014; Gomes & Mendes, 2007; Viera et al., 2017) but used to require that the instructor develop the code for students to use or use other people's materials; the first is time-consuming and the second may not reflect what the instructor wants or needs to do with their students. With ChatGPT and

other LLM's, creating the starting code is simpler for instructors and can be included as part of the assignment so students are interacting with these tools themselves.

The assignments

Students were given two assignments that each used ChatGPT in different ways. The first assignment, "Credit Card Debt," provided students with code that I had generated on ChatGPT; the second assignment, "How Many Zeros," asked students to use ChatGPT themselves to generate code (Jacques, 2023). For each assignment, students were to represent the generated code with either a flowchart or pseudocode then use their representation to determine if the generated code actually solved the given problem. With "Credit Card Debt," students were then tasked with modifying the code to solve a different but related problem. With "How Many Zeros," students needed to develop an alternative approach to solving the same problem then time each approach to determine which was faster.

Participants and Methods

This research project spanned two semesters and three sections of an introductory computer science course taught by two different instructors with similar teaching experience. The first phase occurred in spring, 2023, with 19 undergraduates (8 F / 11 M) at a private university in southeastern USA who had no prior programming experience before being enrolled in an introductory computer science course; the second phase occurred in the same institution during fall, 2023, with 53 undergraduates (17 F / 36 M) who also had no prior programming experience. At the start of data collection, participants had been programming in Python for approximately ten weeks. Participants were given each of the assignments to complete on their own. With each assignment, they were asked to explain the AI-generated code using a flowchart or pseudocode then to create their own code in Python to either modify the generated code (*Credit Card Debt*) or to develop an alternate approach to the problem (*How Many Zeros*). Flowcharts/pseudocode and Python files were collected electronically and de-identified. A qualitative analysis was then performed on all submitted materials. With the flowcharts and pseudocode, analysis identified what processes participants included and if the logic reflected the AI-generated code. With the Python programs, analysis identified any errors or misconceptions participants displayed in their code.

Findings

Most participants chose to create flowcharts rather than pseudocode. With *Credit Card Debt*, only nine participants created pseudocode; with *How Many Zeros* no participants created pseudocode.

In *Credit Card Debt*, eight of the nine who used pseudocode accurately represented the AI-generated code (Fig. 1); the only error was that the loop did not include all necessary steps. For the flowcharts, 17 were accurate representations, 14 used one or more incorrect symbols, 15 did not include the print statements in the loop (Fig. 2), and 17 had both errors. For the coding part of the assignment, all were able to adjust the program to determine how many months would be needed to pay the debt in full. Thirteen also included code to warn the user if their monthly payment was less than the monthly interest.

With *How Many Zeros*, 23 of the flowcharts were correct, seven of which had AI-generated code that counted the zeros using integer arithmetic. The most common errors were forgetting to include the loop from 1 to the user input (n = 19) and other logic errors (n = 10). Other errors included not breaking down the steps correctly (n = 7) or using symbols incorrectly (Fig. 4) (n = 5). Eight had multiple errors (e.g., Fig. 3). Forty-three of the programs correctly created an alternative subroutine to solve the same problem. The remaining ten treated the number as a string in both the AI-generated code and their own (Fig. 5). There were more errors of this kind in the spring semester than in the following fall because the researcher slightly changed the wording on this assignment for the second iteration: Instead of referring to the numeric version as the integer approach, the assignment called it the arithmetic approach.

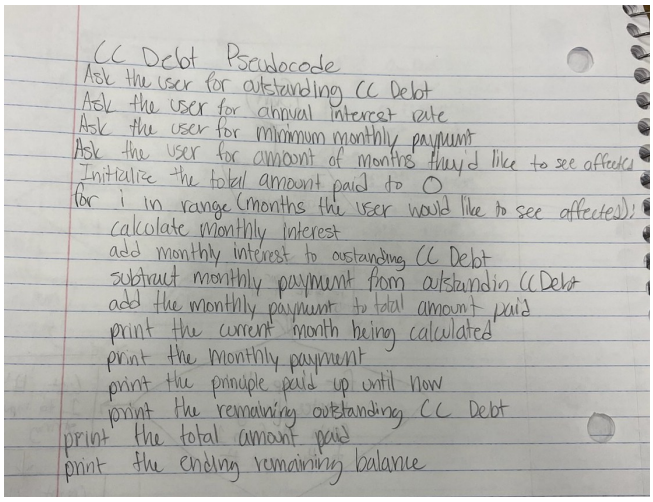
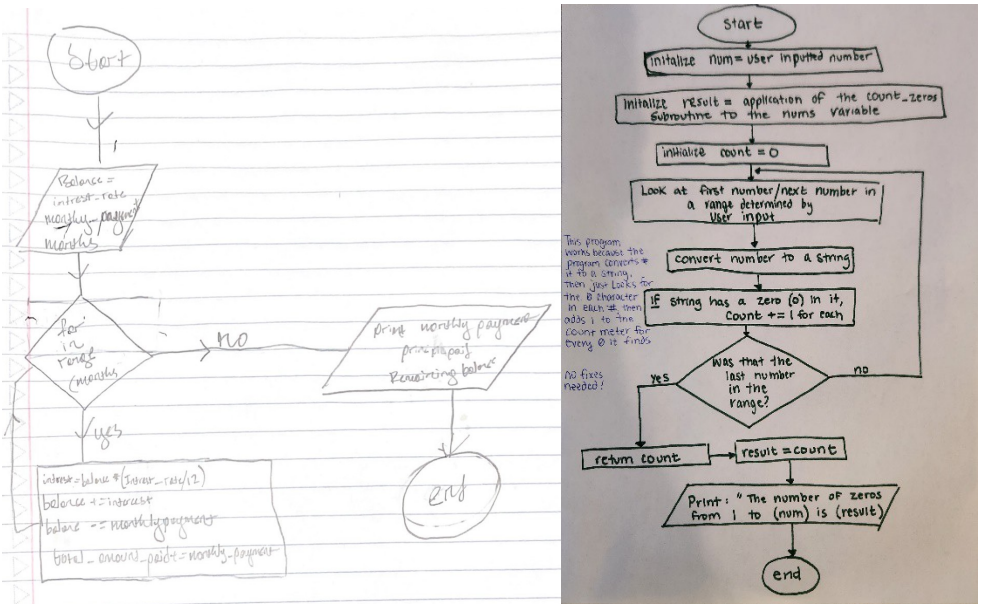
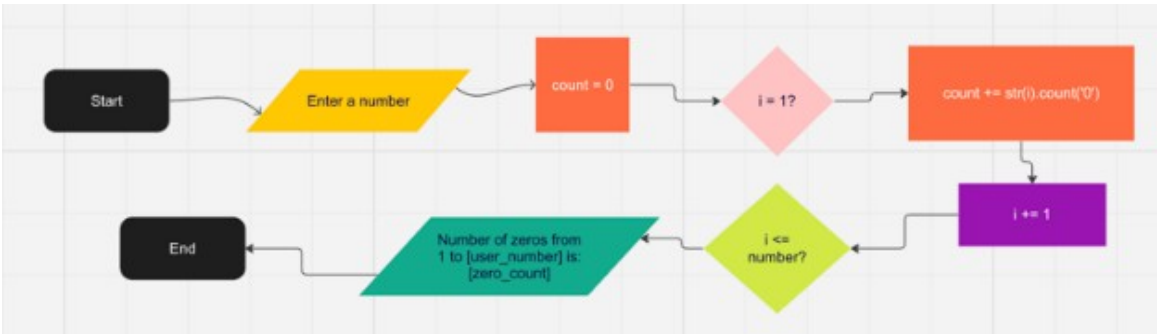


Figure 1: Pseudocode accurately representing AI-generated code.





Figures 2, 3, 4: Flowcharts inaccurately representing AI-generated code.

```python
#This subroutine keeps the input as an integer.

def count_zeros(n):
    count = 0
    for i in range(1, n+1):
        count += str(i).count('0')
    return count

number = int(input("Enter a number: "))
zeros = count_zeros(number)


print("The number of zeros from 1 to", number, "is :", zeros)


#This subroutine casts the input as a string.

def count_zeros(num):
    start_time = time.time()
    num_str = str(num)
    count = 0
    for digit in num_str:
        if digit == '0':
            count += 1
    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"The number of zeros from 1 to {num} is : {count}")
    print(f"Elapsed time: {elapsed_time:.6f} seconds")

user_input = input("Enter a number: ")
```

Figure 5: ChatGPT-generated subroutine and student-created subroutine.

Discussion

In this study, the student work revealed that many participants had difficulties accurately representing an algorithm written in Python. Some of the issues were trivial, such as using incorrect symbols in a flowchart, but other errors suggest that beginning programmers have difficulties describing the logical flow of a program. This study also suggests that they are able to modify code to solve a different problem but have difficulties when asked to develop an alternative algorithm for the same problem. Additionally, this study suggests that beginning programmers may not recognize what variable type a value is being treated as in a Python program, since Python does not make that explicit like other languages do, and so instructions need to indicate how the values are being manipulated and not what their type is in order to have students accurately recognize what approach the LLM generated.

This study has immediate implications for instructors. Having students use LLM's to generate code quickly provides them with something they can use, allowing instructors to increase activities designed for critical thinking. Doing so has the added benefit of informing the instructor what concepts or skills have resonated with students and which need further instruction. Most of the findings in this study are likely language-independent, but it would be interesting to learn if the variable-type difficulties also occur in an introduction CS course that uses a programming language with strict variable typing, such as Java. It would be interesting to learn what other instructors develop for LLM-enabled assignments and what student misconceptions are revealed from them.

References

Aydin, E. (2005). The use of computers in mathematics education: A paradigm shift from "computer aided instruction" towards "student programming." *The Turkish Online Journal of Educational Technology, 4*(2).

Becker, B., Craig, M., Denny, P., Keuning, H., Kiesler, N., Leinonen, J., Luxton-Reilly, A., Malmi, L., Prather, J., and Quille, K. (2023). Generative AI in Introductory Programming. In *Computer Science Curricula 2023, Curricular Practices Volume*. ACM, New York, NY, USA

Blackwell, A.F., Whitley, K.N., Good, J., and Petre, M. (2001). Cognitive factors in programming with diagrams. *Artificial Intelligence Review, 15*(1), 95-114.

Corney, M., Fitzgerald, S., Hanks, B., Lister, R., McCauley, R., and Murphy, L. (2014). 'Explain in plain English' questions revisited: Data structures problems. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, 591-596.

Gomes, A., and Mendes, A. J. (2007, September). Learning to program-difficulties and solutions. In *International Conference on Engineering Education–ICEE*, 7.

Harel, I., and Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments, 1*, 1-32.

Jacques, L. (2023). Teaching CS-101 at the dawn of ChatGPT. *ACM Inroads, 14*(2), 40-46.

Kafai, Y. (1996). Software by kids for kids. *Communications of the ACM, 39*(4), 38-39.

Khalid, M., Saad, S., Hamid, S.R.A., Abdullah, M.R., Ibrahim, H., and Shahrill, M. (2020). Enhancing creativity and problem solving skills through creative problem solving in teaching mathematics. *Creativity Studies, 13*(2), 270-291.

Kolakowski, N. (2023, June 30). Are developers actually using A.I. tools in their workflows? *Dice.com: Career Advice*. Retrieved from https://www.dice.com/career-advice/are-developers-actually-using-a.i.-tools-in-their-workflows.

Laursen, S. L., and Rasmussen, C. (2019). I on the prize: Inquiry approaches in undergraduate mathematics. *International Journal of Research in Undergraduate Mathematics Education, 5* , 129-146.

(NCTM) National Council of Teachers of Mathematics. (2014). *Principles to Actions: Ensuring Mathematical Success for All.* Reston, VA: NCTM.

(NRC) National Research Council. (2001). *Adding It Up: Helping Children Learn Mathematics*. Washington, DC: The National Academies Press. Retrieved from https://doi.org/10.17226/9822.

Prather, J., Denny, P., Leinonen, J., Becker, B. A., Albluwi, I., Craig, M., Keuning, H., Keisler, N., Kohn, T., Luxton-Reilly, A., MacNeil, S., Petersen, A., Pettit, R., Reeves, B., and Savelka, J. (2023). The robots are here: Navigating the generative ai revolution in computing education. In Proceedings of the *2023 Working Group Reports on Innovation and Technology in Computer Science Education*, 108-159.

Rich, P., Bly, N., and Leatham, K. (2014). Beyond cognitive increase: Investigating the influence of computer programming on perception and application of mathematical skills. *Journal of Computers in Mathematics and Science Teaching, 33*(1), 103-128.

Rittle-Johnson, B, and Star, J.R. (2007). Does comparing solution methods facilitate conceptual and procedural knowledge? An experimental study on learning to solve equations. *Journal of Educational Psychology, 99*(3), 561-574.

Vieira, C., Magana, A., Falk, M., and Garcia, R. (2017). Writing in-code comments to self-explain in computational science and engineering education. *ACM Transactions on Computing Education, 17*(4), 1-21. https://doi.org/10.1145/3058751

Wing, J. M. (2006). *Computational Thinking*. ACM: New York. doi:10.1145/1118178.1118215

Zastudil, C., Rogalska, M., Kapp, C., Vaughn, J., & MacNeil, S. (2023, October). Generative AI in computing education: Perspectives of students and instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*, 1-9.