RESEARCH-ARTICLE

Exploring Undergraduate Computing Tutors' Pedagogical Practices

ESSE CIEGO, University of Florida, Gainesville, FL, United States

SKYLER STEIERT, University of Florida, Gainesville, FL, United States

AMANPREET KAPOOR, University of Florida, Gainesville, FL, United States

PDF Download
3770762.3772542.pdf
23 February 2026
Total Citations: 0
Total Downloads: 65

Published: 18 February 2026

Citation in BibTeX format

SIGCSE TS 2026:The 57th ACM Technical Symposium on Computer Science Education
February 18 - 21, 2026
MO, St. Louis, USA

Conference Sponsors:
SIGCSE

Exploring Undergraduate Computing Tutors' Pedagogical Practices

Esse Ciego
ciegosteve@ufl.edu
University of Florida
Gainesville, Florida, USA

Skyler Steiert
steiertskyler@ufl.edu
University of Florida
Gainesville, Florida, USA

Amanpreet Kapoor
kapooramanpreet@ufl.edu
University of Florida
Gainesville, Florida, USA

Abstract

With the rapid increase in students majoring in computing, undergraduate teaching assistants (UTAs) have become a key resource for managing large courses and assisting students. Understanding the pedagogical practices employed by UTAs is essential for improving tutoring quality and meeting student learning needs. Our research aims to explore what pedagogical practices UTAs use during office hours and when they use them. To investigate these practices, we conducted an ethnographically informed study with seven UTAs, observing them for 37.5 hours in 52 tutoring sessions during office hours and documenting their interactions with students. Our study revealed that UTAs employ six pedagogical practices: *asking open-ended questions, hands-on demonstrating, prompting, referencing resources, explaining,* and *expressing affect*. Tutors tended to rely on explanations, questions, and references to resources to teach concepts, as well as hands-on demonstrations and expressions of affect to debug. Our findings highlight key tutoring strategies that can inform future training and support for UTAs.

CCS Concepts

• **Social and professional topics → Computing education**.

Keywords

peer tutoring, undergraduate teaching assistant, pedagogy

ACM Reference Format:
Esse Ciego, Skyler Steiert, and Amanpreet Kapoor. 2026. Exploring Undergraduate Computing Tutors' Pedagogical Practices. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2026), February 18–21, 2026, St. Louis, MO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770762.3772542

1 Introduction

With the rapid rise in enrollment of undergraduate computing majors [3], many CS departments are employing undergraduate teaching assistants (UTAs) to help manage the increased workload with large class sizes. As such, UTAs have become a widely used resource for students. These UTAs frequently interact with students by assisting them with their programming assignments in labs and office hours, as well as leading discussion sections to review course material [15]. CS departments with UTA programs report many benefits to their students, such as better student performance [2, 19] and improved satisfaction in courses [5]. Additionally, UTAs are often students' first point of contact with course staff, which can leave a lasting impression on their sense of belonging [18].

UTAs have a lot of autonomy in teaching course content [21], allowing them to personalize students' learning in office hours [6]. However, this flexibility in office hours can be a double-edged sword: interactions with UTAs could either lead students to view them as an excellent resource or as inadequate, unable to provide guidance on conceptual information and problem-solving skills necessary for success [18]. For example, researchers have found that UTAs often give students direct answers to their programming assignments instead of teaching them problem-solving processes [10]. Additionally, work by Molina et al. [25] explains how factors such as queue size and lack of student preparation can hinder tutors' effectiveness. While these studies explore why tutors use certain practices, we know little about what pedagogical strategies UTAs actually use and find effective in specific scenarios [22]. Understanding these pedagogical strategies can offer practical guidance to UTAs for better supporting their students.

To better understand what pedagogical practices UTAs employ, we explored tutors' behaviors through the lens of pedagogy, which Falkner and Sheard define as "the science of how instructors promote learning" through learning activities, strategies, and techniques [7]. The central research question we aim to answer in this paper is: *What pedagogical practices do undergraduate computing tutors use in tutoring sessions and when do they use these practices?*. We use the term *pedagogical practices* to describe the strategies computing tutors use to help students during office hours. To answer our research question, we designed an ethnographically informed study in which we observed seven UTAs at a large public university in the US during office hours with students. These UTAs had varied backgrounds in terms of the courses they tutored for. Observations occurred during their office hours, and we documented their interactions with students and the practices employed. We analyzed our notes inductively to identify an exhaustive list of practices.

Our work highlights how tutors support students during office hours and where they often struggle. These insights can inform UTA training, help tutors reflect on their approaches, and strengthen their confidence in guiding students through debugging.

2 Related Work

In this section, we review prior research on UTAs' pedagogical practices and summarize these practices in Table 1.

Interview studies. In interviews with tutors about their interactions with students, tutors often described taking a "hands-off" approach that encourages their students to be independent learners. Along this line, tutors in Riese and Kann's study reported guiding students to the correct solution through questions about the student's issue, code, and error messages [21]. They recognized the importance of helping students understand underlying processes and conceptual information, as well as making students an active part of figuring out solutions [18, 25]. Their approach explains why some tutors might feel strongly about not touching a student's keyboard [13, 17, 21, 22]. In the rare occasion that tutors described going against best practices, they cited external stressors such as a large queue and students' lack of preparation [25], as well as internal stressors such as a limited understanding of the student's problem [22, 25]. These studies offer insight into tutors' intentions behind their approaches. However, they do not include a detailed overview of their practices, which our work aims to provide.

Observational and ethnographic studies. Observational studies have shown that tutors often go against best practices than expected from interview results. Malysheva et al.'s [12] analysis of recorded virtual office-hours sessions found tutors take a more "hands-on" approach to fix students' code directly. Tutors often walked students through tutors' past solutions, typically reading from their own code. In another study, Kraus-Levy et al. found that in 80% of tutor-student interactions involving code errors, tutors would not help students debug or mention debugging [10]. The differing results between interview and observational studies may stem from their reluctance to admit discouraged teaching strategies in front of their employers. While studies that rely solely on instructor interviews may underreport these practices, our use of peer-level observers conducting observations may have allowed tutors to act more naturally.

Regarding ethnographic investigations, Markel and Guo's autoethnography [13] provides a more nuanced perspective on tutors' pedagogical practices as it contains both descriptions of tutors' teaching strategies and explanations of why tutors used specific strategies. Similar to this work, we use ethnographic methods to investigate how and why tutors teach in office hours. Unlike Markel and Guo's work that focused on only one TA, our study observed multiple tutors from different courses, experience levels, and backgrounds.

Pedagogical practices	Citing articles
Guide student to find solution for themselves	[10, 13, 21, 25]
Trace and examine code for bugs	[10, 13, 25]
Get clarification about student's code or issue	[10, 21]
Directly provide solution to problem	[10, 13, 25]
Reference class or web resources	[21, 25]
Explain concepts	[18, 25]
Compile and run student's program	[10, 13]

Table 1: Commonly reported pedagogical practices of tutors

3 Methods

3.1 Study Design and Research Questions

We designed an ethnographically informed study to answer the following research question: *What pedagogical practices do undergraduate computing tutors use in tutoring sessions and when do they use these practices?* Ethnography is a qualitative research methodology that seeks to understand the values, beliefs, behaviors, and language of a culture-sharing group [9]. Researchers immerse themselves in the daily lives of their participants through extensive observation and interviews. Although ethnography originates from cultural anthropology [4], it has been used in CS education to understand classroom behaviors [1] and UTA's thought processes [13].

Our study aimed to conduct an ethnographically informed investigation with an *etic* focus on undergraduate computing tutors' pedagogical practices. In ethnography, an etic perspective refers to an outsider's analytical viewpoint, emphasizing objective observations rather than participants' internal understandings (*emic* perspective) [4]. For this etic perspective, two researchers observed tutors' pedagogical practices. In addition, the researchers also conducted a survey and semi-structured interviews with tutors and students as part of our larger study. However, for this paper, we focus on analyzing the observational notes data. The institutional ethics review board approved our study at our university under the exempt category.

3.2 Research Site, Population, and Sample

We conducted the study in the CS department of a large public university in the United States. Our study population consists of UTAs supporting students in core computing courses (CS1 and CS2), with 500 to 800 students enrolled per semester. We recruited tutors at our institution who are either (a) employed by course instructors as peer mentors (UTAs) to run office hours, lab sessions, or discussion sections for introductory undergraduate computing courses or (b) employed as tutors in the department tutoring center. Instructors at our university vary in their selection processes of UTAs and have autonomy to recruit UTAs based on enrollments in their respective courses. They commonly evaluate UTA applicants on their performance in the course and whether they have a minimum GPA of 3.4. Additionally, UTAs must take a CS teaching and learning course during their first semester of tutoring, which covers UTA training and educational strategies for working with students from diverse backgrounds.

3.3 Tutor and Student Recruitment

3.3.1 Tutor. To recruit tutors for our study, we asked six core computing course instructors to present our research goal at their teaching staff meeting. These courses consisted of eight to 23 tutors. Seven tutors consented to participate in our study and provided their availability in an online form. Instructors were not informed of specific tutors' involvement in the study. A similar approach was made to recruit tutors from the CS department's tutoring center, with researchers asking the manager of the tutoring center. Six tutors staff the center, serving about 80 students per semester. We offered no incentive for UTAs to participate in the study.

3.3.2 Student. During office hours, students were asked for their consent to allow researchers to observe them and the tutor during the session. To recruit students for the interviews and survey, students were offered one percent extra credit for their participation, contingent on their respective course instructors. However, only one student participated in the survey.

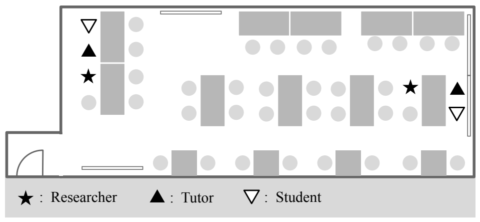

Figure 1: Example seating arrangement in observation site

3.4 Data Collection

3.4.1 Logistics. We collected data in the fall of 2024 by observing seven UTAs from introductory programming courses (Programming Fundamentals (Prog. Fund.) 1 & 2, Advanced Programming Fundamentals, Data Structures Algorithms (DSA)) and over 10 days during their office hours between October 25th to November 29th, 2024. Data collection primarily occurred in a natural setting: a public, enclosed room in the building of the CS Department (see Figure 1). The room held tutors who wanted their office hours to be in-person or hybrid.

For data organization purposes, we collected information by session. Sessions are defined as interactions during the tutor's office hours between the tutor and one student. Every time a student attended office hours, we documented their interaction with the tutor in a session. Using seven tutors and 37.5 hours of observation, we collected data in 52 sessions (see Table 2).

# Tutor	Course	Hrs Observed	# of Sess.	Modality	Observer
1	Advanced Prog. Fund.	9.0	16	IP + V	A1
2	Prog. Fund. 2 & DSA	7.5	10	IP	A1 + A2
3	Prog. Fund. 2 + DSA	2.0	3	IP	A1 + A2
4	All CS intro.	1.0	1	V	A1
5	DSA	7.0	3	IP + V	A1 + A2
6	Prog. Fund. 1	9.0	18	IP + V	A1 + A2
7	Prog. Fund. 1	2.0	1	V	A2

S = Students; IP = In-person; V = Virtual; A1, A2 = First and Second Authors;

Table 2: Tutor Observations Summary

Data was collected through observations, semi-structured interviews, and qualitative surveys. However, for this paper, we focus on analyzing the observational data. For these observations, two researchers took semi-structured notes on a spreadsheet application. Each row contained data for a session between a tutor and a student. The sheet's columns contained basic information, such as participant details, description of observation settings, and time markers. The sheet also had descriptions of the tutors' and students' behavior, relevant quotes from those tutors, notes of tutors' responses in open interviews, and reflective memos.

The first author drafted initial categories for the observation sheet. The research team later refined the observation sheet based on two practice runs observing tutors and students. During observations, researchers focused on the tutors' actions and the resources tutors used or referenced. To ensure the privacy of the tutors and students, the researchers concealed their names and other identifiable information in the notes using pseudonyms.

3.5 Data Analysis and Interpretation

After observing a tutoring session, the researchers wrote memos to document their thought-process [16]. These memos described initial categories for tutoring strategies and compared tutors' behaviors towards their student. They then used inductive content analysis [14] to code qualitative data, starting with open coding and later abstracting codes into six categories of pedagogical practices.

3.6 Positionality

A researcher's positionality influences many aspects of a qualitative study, such as the chosen topic, methodology, and results [8, 11, 23]. As the first two authors observed and analyzed the data, they practiced reflexivity through memos and team discussions.

The first author is a third-year undergraduate CS major at a large public US university. Lacking experience leading sections as a general CS tutor, they were biased toward tutors who emphasized conceptual understanding and used explaining strategies.

The second author is a second-year undergraduate CS major at the same university. Though initially inexperienced in tutoring, they were motivated by a strong belief in the value of supportive, student-centered teaching. Their background and reliance on tutors in prior coursework informed a bias toward open-minded tutors with varied strategies.

The last author is a CS Education researcher and teaching faculty who supervised the study. He values active learning techniques, guided scaffolding, and use of educational tools. His role as an educator and researcher may have influenced data interpretation.

4 Results

We found six categories of pedagogical practices from our observation of tutors' behavior using inductive data analysis. These categories emerged from 35 unique codes. In this section, we describe each practice and code, provide examples for context, and report its frequency across the sessions.

4.1 Explaining

For this category, tutors used various explaining strategies tailored to students' specific learning needs. They used these strategies to address students' gaps in understanding, clarify misconceptions, and break down complex topics. Explaining was observed in 42 out of 52 sessions and emerged from nine codes.

Explaining program behavior (36/52): In this code, tutors explained to students how their code was behaving or how their code would behave with an implemented change. Tutors utilized this method the most frequently out of all the explanation methods. For example, *T5 explains how student's implementation of page rank algorithm stores ranks in edge, thus not updating each node.*

Explaining code implementation (27/52): Tutors explained how to build, decompose, or structure code for a program in this code. They typically used this practice when students understood the foundational concepts but struggled to translate them into working code. For example, *"Then you're going to put it into test 1, making it now your part 1 function" (T1).*

Explaining concepts (13/52): Tutors explained concepts relevant to the session, ranging from programming language features to general CS topics. They employed this strategy when students struggled to understand the concepts. These ranged from trivial to complex, and could be related to the coding language, assignment specification, or general CS concepts. For example, T5 explained to a student who was preparing for a test and did not understand a substring matching algorithm that *"the idea is that you are turning every character into binary code for every time it's used"*.

Explaining the tutors' debugging process (8/52): Tutors explained how to use a type of debugging tool for students to understand how their code is behaving. This code explains how to use the debugger (either the IDE debugger or print statements) so that students can identify issues on their own in the future. For example, *T1 explained to a student how using the debug button in their IDE and adding a "breakpoint" will stop at that line.*

Explaining using visual diagrams (7/52): Tutors drew a visual representation of concepts for students to understand better. Tutors used paper and pencil, physical whiteboards, or an online drawing program like Microsoft Paint or Zoom's annotation tool. Tutors used this strategy when a concept was difficult to visualize, such as in the Data Structures course. For example, *T2 attempted to imply a different starting index than zero for a student's assignment. After unsuccessful attempts at other explaining methods, T2 used pen and paper to draw out how the program should behave.*

Explaining using examples (6/52): Tutors provided examples alongside their explanation to help a student understand concepts. For example, *T6 uses provided examples from assignment specifications to explain assignment workflow.*

Explaining development environment tools (4/52): Tutors explained features and configurations of the students' development environments, such as compilers or environment variables. For example, *T3 explains that the student's installation uses UCRT (Universal C Runtime library), which is the runtime they would get when installing via Chocolatey.*

Explaining using metaphors (3/52): Tutors utilized a metaphor or analogy alongside a concept to better communicate the concept. They used this practice when other means of explanation or questioning with the students failed. For example, *T1 compares recursion to asking neighbors (and their neighbors' neighbors) for sugar.*

Explaining with pseudo-code (2/52): Tutors created or referenced existing pseudo-code to explain a concept that the student was confused about. For e.g., *when a student did not know how to reinterpret a structure as a character array, T1 drew out ((char*)&struct).*

4.2 Questioning

In this category, tutors frequently ask open-ended questions to engage students and gauge their understanding. Tutors also used questioning to get students to explain their reasoning verbally. Tutors used questioning at least once in 41 of the 52 sessions and this category emerged from six codes.

Asking checking questions (21/52 sessions): In this code, tutors asked if students had any issues or questions. Tutors used these questions to initiate conversation and see whether students had additional issues. For example, *T6 often asked "Do you have any questions?" at the end of each of session.*

Asking tracing questions (16/52): Tutors asked students to reason about the program's state, ranging from general questions on program behavior to more specific questions about the value of variables. Tutors used these questions to help students better understand the behavior of their current program. For example, *T2 asked the student to explain their program logic with "If I cout [console output] blue string, what is it going to output?".*

Asking conceptual questions (10/52): Tutors asked students to recall facts about the programming language or concepts from the lecture. For example, *"What's the size of int?" (T1).*

Asking navigation questions (7/52): In this code, tutors asked students to navigate to a location of the code, such as a variable, function, or class. For example, *"Where does that flat data come from?" (T6), "Do you have a function to get the texture of a sprite?" (T2),* and *"Where does flat data get stored in your program?" (T6).*

Asking progress questions (7/52): Tutors asked students about their current progress on the assignment, usually at the start of the session. For example, *"How many hours have you spent on the project?" (T1),* and *"What did you get on the project?" (T3).*

Asking resource questions (1/52): One tutor asked what resources student referenced for their assignment. For example, *"Have you used ChatGPT?"(T2).*

4.3 Prompting

In this category, the tutors *instructed* students to make specific changes to their code or development environment. Unlike hands-on demonstrating (see Section 4.6), prompting was paired with various other strategies, allowing students more autonomy. This category was observed in 35 of the 52 sessions and it emerged from five codes.

Prompting to edit code (23/52): Tutors prompted or instructed students to edit their code logic. This code included statements or questions that implied students' current code was incorrect. Tutors used this approach to guide students' thinking and subtly correct code implementations. For example, *T2 asks "Can you multiply 255.0 and add 255?" to prompt the student to revise their image-scaling formula.*

Prompting to run the program (17/52): In this code, tutors prompted students to run code in their IDE or submit it to an autograder. For example, *After suggesting to write another else branch in their program, T asks student to resubmit their program to the autograder.*

Prompting to run debugger (8/52): Tutors told students to run the debugger or to add print-debug statements. This strategy encouraged students to understand runtime behavior. For example, *T4 instructed students to print out program variable states.*

Prompting to navigate code (7/52): Tutors guided students to specific parts of the code, such as functions or classes, to find relevant logic or spot issues.

Prompting to edit development environment setup (3/52): Tutors instructed students to make changes to their development environment when they had issues with IDE settings or installing software. Tutors often prompted students after identifying their setup as a blocker for the task. For example, *"Can you install SFML (Simple and Fast Multimedia Library) and check if it works?" (T1).*

4.4 Expressing Affect

In this category, tutors used language to influence students' emotions. These expressions of affect reassured, encouraged, and guided students through challenges faced during the session to create a positive and productive tutoring experience. Expressing affect was observed in 31 of the 52 sessions, emerging from seven codes.

Expressing certainty (11/52): Tutors expressed confidence in the correctness of students' code during their sessions. They used this method to affirm the success of the code changes, whether they were made by students or tutors themselves. For instance, *after a student edited their code based on T2's suggestion, T2 assured them that their process is correct and they are "almost there."*

Affirming student independence (11/52): Tutors urged students to work independently by withdrawing assistance, either momentarily or for the rest of the session. Tutors used this method when students showed enough understanding to work independently. For example,*T1 encouraged a student to debug their program individually after explaining how to use diffchecker.*

Expressing lack of knowledge (10/52): Tutors expressed that they had incomplete knowledge about a concept, function, or assignment. Tutors used this method when they feared their answers may be incorrect, potentially leading students in the wrong direction. For example, *"I don't remember if this is a forward slash or a backslash" (T2).*

Expressing confusion (7/52): Tutors expressed confusion about students' program behaviors, typically when they did not initially catch a student's bug. For example, *a student's program runs but outputs a black screen, prompting T2 to express "That's weird"* .

Expressing empathy, regret, or concern (7/52): Tutors expressed empathy, regret, or concern about students' issues. Tutors used this method when they believed that they could not provide much help in solving the students' problems. For example, *T1 said they "feel bad" about a student's CMakefile not being able to compile, as they would get a poor grade.*

Expressing early warning (6/52): Tutors warned the students about the difficulty or complexity of an assignment or encouraged them to begin the project early and not to procrastinate. Tutors used this method to warn students about the difficulty of an assignment, drawing on previous experience from the course. For example, *"It's hard to get started, but once you get started, it gets easier" (T1).*

Expressing praise (2/52): Tutors expressed appreciation or acknowledgment of a student's effort, response, or progress in this code. For example, *"We are winning this!" (T2).*

4.5 Referencing Resources

In this strategy, tutors encouraged students to consult course-specific materials, documentation, or online forums like StackOverflow. This practice supported students in independently seeking information, particularly from resources recommended by the course instructor. We observed this behavior in 28 of the 52 sessions, emerging from four codes.

Referencing course-specific (internal) sources (18/52): Tutors told students to refer to course resources such as assignment specifications, Canvas modules, documents, and course-designated Discord channels. Tutors commonly mentioned course specifications, discussed examples, and program requirements. For example, *T1 asks a student, "Do you keep track of the [course] Discord?" to direct them to where project details and support are posted.*

Referencing tools (4/52): Tutors recommended that students use coding tools for debugging or testing. *"This is why I always tell my students to run a hex editor" (T1),* and *T1 asks students to use a diff checker.*

Referencing external sources (3/52): Tutors referred students to resources not managed by the course instructors or TAs, most commonly StackOverflow. For example, *T2 recommended watching YouTube videos to learn how to use text editors*, and *T1 provided StackOverflow links on how to use a text comparison application.*

Referencing human help (1/52): Tutors referred students to other TAs or course instructors to ask questions or get live support. For example, *before T6's office hours ended, T6 recommended a student go to the instructor's office hour, which were immediately afterward.*

4.6 Hands-on demonstrating

In this category, tutors fixed challenging issues by editing students' code or development environment directly on their laptops. Then, they followed up with other hands-on demonstration strategies, which resulted in taking a more active role in the session. We observed hands-on directing at least once in 16 of the 52 sessions and categorized it into four codes.

Hands-on navigation (9/52): Tutors used students' keyboards or mice to scan through their codebase, typically to find issues quickly or better understand their program structure.

Hands-on editing (6/52): Tutors directly edited students' code to make minor fixes or help implement suggestions the student had difficulty applying.

Hands-on program running (6/52): Tutors ran students' programs to observe their behavior.

Hands-on debugging (4/52): In this code, tutors ran the students' programs in debug mode. Tutors created breakpoints, stepped through execution, and looked at the states of variables. Tutors directly debugged involved, complicated projects and identified subtle bugs. For example, *T2 printed "hello" in a method's definition to check if it was running.*

5 Discussion

5.1 Tutors' pedagogical strengths & challenges

5.1.1 Tutors excel at guiding implementation. We observed tutors use many pedagogical practices in one session, especially when helping students get started on their programming assignments. They frequently combined technical explanations of how to implement their program with whiteboard diagrams, references to assignment specifications, and questions that ask students to trace through program execution. Tutors might be comfortable explaining due to practice from teaching in discussion sections, where they might use similar pedagogical practices. Our observations align with interview studies where tutors emphasize helping students understand concepts and problem-solving processes [18, 25].

However, our work diverges from prior work in terms of tutors' approach to explanation. While Malysheva et al. found that many tutors read off their implementation to the students [12], our UTAs attempted to build students' conceptual understanding by explaining concepts. In most instances, tutors we observed could describe how to implement a solution without referring to their code. The difference between our results and those of Malysheva et al. is likely because we observed self-selected tutors in introductory programming courses who might be motivated while they studied tutors for a more advanced programming course.

Building on this, we also offer a more specific view of how tutors use questioning to support learning. For instance, Krause-Levy et al. categorized questions that led students to an answer as "guiding" [10] while Molina et al. defined guiding questions more generally as those that help students make progress on their assignment [25]. Our findings provide a more specific account: effective guiding questions either prompted students to recall relevant concepts or, more commonly, helped them see how their current program behaves and how the intended solution should behave. Additionally, while Markel and Guo's model depicted tutors trying to understand students' code early in the session [10], we found tutors' efforts to understand persisted throughout the session and evolved in response to a student's needs and progress.

5.1.2 Tutors struggle with guiding through debugging. Most tutors adopted a hands-on or prompting-based approach to debugging. We commonly saw tutors repeatedly follow this sequence of practices: tutors would directly or prompt students to (1) navigate their codebase, (2) edit their code, (3) run their code, and finally (4) explain the fix or repeat steps 1-3 if the fix was not successful. This approach appears to contradict commonly reported best practices by tutors, such as avoiding direct interaction with a student's laptop or refraining from giving immediate answers [13, 17, 22, 25]. Additionally, we observed that for complex projects or issues with a student's development environment, tutors were more likely to use hands-on demonstrations. This reveals a tension between tutors trying to understand a student's problem and taking over. These issues highlight that tutors need to be trained in teaching students how to debug by using a more Socratic approach or by showing students how to approach debugging systematically.

Tutors' emotional responses during debugging influenced their approach. They often expressed confidence in fixing "simple" bugs, justifying stepping in quickly. However, when assignments behaved unexpectedly, especially after tutors had expressed confidence, they tended to rely more on hands-on demonstration. This reflects how unexpected challenges increased their urge to take control, limiting student-led problem-solving. Molina et al. [25] note that tutors' limited content knowledge can also hinder effective office hours, which may contribute to this hands-on approach.

5.2 Recommendations for tutors

5.2.1 Reflect on pedagogical practices. To shift their tutoring habits, UTAs should reflect on their current pedagogical practices. Similar to the methodology in Ren et al., tutors can complete brief exit surveys to document students' issues [20] and record their own practices to solve the encountered issues. To assess student learning, tutors can analyze how engaged students were and how often they reflected (similar to Sanders et al. [24]). Tutors may also benefit from discussing their approaches with other tutors, such as sharing ways to explain difficult topics and what strategies worked in specific contexts. In any case, tutors can use the code book from our study to have a consistent language to describe their practices.

5.2.2 Use a structured pedagogical debugging approach. Tutors should try to reduce hands-on demonstration strategies when helping students debug their code and adopt a systematic, prompt, or question-based approach. This approach should guide students through debugging, asking them: *Why is this bug occurring? What do you think is causing it? Can you make a prediction and test it?* [27]. These questioning strategies help students verbalize their reasoning and take the lead in the session. Since this approach is tool- and language-agnostic, tutors and students can apply it across different courses and programming environments. To further support tutors, CS departments can include debugging modules in existing UTA training courses or offer a dedicated elective course on debugging (as suggested by Wilkin [26]), which instructors can recommend to tutors seeking to improve their skills.

5.3 Limitations

Our study faced several challenges and limitations. First, the researchers faced occasional technical issues when observing virtual office hours. As they did not have direct access to the communication platforms used for these sessions, such as Zoom or Discord video calls, they were sometimes unable to observe tutor-student interactions fully.

Second, our qualitative analysis may limit generalizability, is open to multiple interpretations, and may have been influenced by the researchers' presence. To address these limitations, we conducted repeated observations with a tutors to build rapport and reduce observer effects. Additionally, we improve the reliability of our research process through transparency, disclosure of our positionality, a clear definition of our study's scope, and the inclusion of rich participant quotes to strengthen the validity of our analysis.

Lastly, UTAs who participated in our study were self-selected, which may have biased the sample toward more motivated or confident UTAs and may not represent the broader population.

6 Conclusion

Our work expands on earlier ethnographic and observational work by examining tutoring through the eyes of both tutors and students, providing a grounded, insider perspective. We found that tutors could explain concepts and support student understanding, drawing from various tutoring techniques. However, they often struggled with debugging, relying on hands-on demonstration approaches with minimal explanation or student engagement in certain cases. Our findings can inform practical, structured tutoring guides usable across contexts.

In the future, researchers could design and evaluate interventions that (1) encourage tutors to reflect on their pedagogical practices and (2) teach tutors how to debug systematically. Additionally, researchers can explore tutoring practices within discussion sections or labs where tutors conduct reviews, recitations, or concept overviews, teaching many students at the same time.

7 Acknowledgments

We thank the tutors and students who participated in our study, as well as the reviewers who provided feedback on our manuscript. [1]

[1]Note that we used ChatGPT and Writefull in Overleaf to verify grammar and reword small portions of the text in this paper.

References

[1] Lecia J Barker and Kathy Garvin-Doxas. 2004. Making visible the behaviors that influence learning environment: A qualitative exploration of computer science classrooms. *Computer Science Education* 14, 2 (2004), 119–145.

[2] Saúl A. Blanco. 2018. Active Learning in a Discrete Mathematics Class. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 828–833. doi:10.1145/3159450.3159604

[3] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: the growth of computer science. *ACM Inroads* 8, 2 (May 2017), 44–50. doi:10.1145/3084362

[4] John W. Creswell and Cheryl N. Poth. 2018. *Qualitative Inquiry and Research Design: Choosing Among Five Approaches*. SAGE Publications, Thousand Oaks, CA, Chapter 4.

[5] Adrienne Decker, Phil Ventura, and Christopher Egert. 2006. Through the looking glass: reflections on using undergraduate teaching assistants in CS1. *SIGCSE Bull.* 38, 1 (March 2006), 46–50. doi:10.1145/1124706.1121358

[6] Paul E. Dickson, Toby Dragon, and Adam Lee. 2017. Using Undergraduate Teaching Assistants in Small Classes. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 165–170. doi:10.1145/3017680.3017725

[7] Katrina Falkner and Judy Sheard. 2019. *Pedagogic Approaches*. Cambridge University Press, 445–480.

[8] Jonathan Grix. 2004. *The Foundations of Research*. Palgrave Macmillan.

[9] Marvin Harris. 2001. *The rise of anthropological theory: A history of theories of culture*. AltaMira Press.

[10] Sophia Krause-Levy, Rachel S Lim, Ismael Villegas Molina, Yingjun Cao, and Leo Porter. 2022. An exploration of student-tutor interactions in computing. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. 435–441.

[11] Kirsti Malterud. 2001. Qualitative research: standards, challenges, and guidelines. *The lancet* 358, 9280 (2001), 483–488.

[12] Yana Malysheva, John Allen, and Caitlin Kelleher. 2022. How Do Teaching Assistants Teach? Characterizing the Interactions Between Students and TAs in a Computer Science Course. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–9. doi:10.1109/VL/HCC53370.2022.9832962

[13] Julia M. Markel and Philip J. Guo. 2021. Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 502–508. doi:10.1145/3408877.3432533

[14] Philipp Mayring. 2000. Qualitative Content Analysis. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 1, 2 (Jun. 2000). doi:10.17169/fqs-1.2.1089

[15] Diba Mirza, Phillip T. Conrad, Christian Lloyd, Ziad Matni, and Arthur Gatin. 2019. Undergraduate Teaching Assistants in Computer Science: A Systematic Literature Review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 31–40. doi:10.1145/3291279.3339422

[16] Phyllis Montgomery and Patricia H Bailey. 2007. Field notes and theoretical memos in grounded theory. *Western Journal of nursing research* 29, 1 (2007), 65–79.

[17] Elizabeth Patitsas. 2013. A case study of the development of CS teaching assistants and their experiences with team teaching. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '13)*. Association for Computing Machinery, New York, NY, USA, 115–124. doi:10.1145/2526968.2526981

[18] Leah Perlmutter, Jean Salac, and Amy J. Ko. 2023. "A field where you will be accepted": Belonging in student and TA interactions in post-secondary CS education. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 356–370. doi:10.1145/3568813.3600128

[19] Inna Pivkina. 2016. Peer learning assistants in undergraduate computer science courses. In *2016 IEEE Frontiers in Education Conference (FIE)*. 1–4. doi:10.1109/FIE.2016.7757658

[20] Yanyan Ren, Shriram Krishnamurthi, and Kathi Fisler. 2019. What Help Do Students Seek in TA Office Hours?. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 41–49. doi:10.1145/3291279.3339418

[21] Emma Riese and Viggo Kann. 2020. Teaching Assistants' Experiences of Tutoring and Assessing in Computer Science Education. In *2020 IEEE Frontiers in Education Conference (FIE)*. 1–9. doi:10.1109/FIE44824.2020.9274245

[22] Emma Riese, Madeleine Lorås, Martin Ukrop, and Tomáš Effenberger. 2021. Challenges faced by teaching assistants in computer science education across europe. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 547–553. doi:10.1145/3430665.3456304

[23] Wendy E Rowe. 2014. Positionality. *The SAGE encyclopedia of action research* 1 (2014), 627–8.

[24] Kate Sanders, Jonas Boustedt, Anna Eckerdal, Robert McCartney, and Carol Zander. 2017. Folk Pedagogy: Nobody Doesn't Like Active Learning. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 145–154. doi:10.1145/3105726.3106192

[25] Ismael Villegas Molina, Jeannie Kim, Audria Montalvo, Apollo Larragoitia, Rachel S. Lim, Philip J. Guo, Sophia Krause-Levy, and Leo Porter. 2025. Undergraduate Computing Tutors' Perceptions of their Roles, Stressors, and Barriers to Effectiveness. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1155–1161. doi:10.1145/3641554.3701784

[26] G. Aaron Wilkin. 2025. "Debugging: From Art to Science" A Case Study on a Debugging Course and Its Impact on Student Performance and Confidence. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1225–1231. doi:10.1145/3641554.3701893

[27] Andreas Zeller. 2005. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.