An Exploration of Student-Tutor Interactions in Computing

Sophia Krause-Levy
University of California San Diego
skrausel@eng.ucsd.edu

Rachel S. Lim
University of California San Diego
ral077@ucsd.edu

Ismael Villegas Molina
University of California San Diego
isvilleg@ucsd.edu

Yingjun Cao
University of California San Diego
yic242@eng.ucsd.edu

Leo Porter
University of California San Diego
leporter@eng.ucsd.edu

ABSTRACT

As enrollments in computing courses have surged, the ratio of students to faculty has risen at many institutions. Along with many other large undergraduate programs, our institution has adapted to this challenge by hiring increasing numbers of undergraduate tutors to help students. In early computing courses, their role at our institution is primarily to help students with their programming assignments. Despite our institution offering a training course for tutors, we are concerned about the quality and nature of these student-tutor interactions. As instruction moved online due to COVID-19, this provided the unique opportunity to record all student-tutor interactions (among consenting participants) for research. In order to gain an understanding of the behaviors common in these interactions, we conducted an initial qualitative analysis using open coding followed by a quantitative analysis on those codes. Overall, we found that students are not generally receiving the instruction we might hope or expect from these sessions. Notably, tutors often simply give students the solution to the problem in their code without teaching them about the process of finding and correcting their own errors. These findings highlight the importance of tutoring sessions for learning in introductory courses and motivate remediation to make these sessions more productive.

CCS CONCEPTS

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

KEYWORDS

undergraduate tutors, tutoring, student outcomes

ACM Reference Format:
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 Innovation and Technology in Computer Science Education Vol 1 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3502718.3524786

1 INTRODUCTION

Computing programs have seen increased enrollments, which has led some institutions to increase the size of their tutoring programs to offer additional one-on-one support to students [23]. Recent work has begun to look into the benefits of undergraduate tutoring programs in computing. The majority of the work focuses on the benefits of the relatability [7, 9, 32], approachability [7, 9, 28, 32] and informality [28] of undergraduate tutors rather than on the content of the interaction itself. However, a recent study began to examine the content and process of student-tutor interactions through the perspective of a single tutor [19].

Given that students generally prefer interactions with course staff (such as tutors) over course instructors [15] and that there are often more contact hours available with tutors than with instructors, it is likely that the majority of the time a student seeks help, they receive that help from a tutor. At our institution, there are often more than ten times as many contact hours available from tutors than from instructors, and tutor hours are heavily attended. There is reason to be optimistic that these tutoring sessions are valuable, given that prior work has shown that one-on-one tutoring sessions are often successful because of the tutor's ability to customize their instruction to the students' needs [1, 20, 21]. However, it is unclear what is happening in these help sessions in computing classes and if they are as effective as we might hope.

A deeper understanding of what is happening during these student-tutor interactions can help identify ways to improve how we teach our tutors and how our tutors, in turn, teach our students. The advent of all instruction being moved online due to COVID-19 provided us with the unique opportunity to study these student-tutor interactions directly by recording the sessions on Zoom. Our dataset thus consists of recorded student-tutor interactions in two courses at our university.

To analyze our videos, we used an open-coding approach to identify and categorize what occurs during these interactions. During the analysis, clear categories emerged based on what can be perceived as effective and ineffective tutoring practices. From the quantitative analysis of these codes, clear trends emerged in the nature of student-tutor interactions of relevance to the community.

Critically, we found that students may not be getting the instruction that one hopes they would receive during these sessions. Specifically, tutors often give students the solutions to their problems without teaching them, or engaging with them in, the process of finding and correcting their own errors. These findings highlight the need for further research on ways to make these sessions more productive for students.

2 RELATED WORK

To our knowledge, we are the first to analyze actual video footage of undergraduate student-tutor interactions during regular tutoring

ITiCSE 2022, July 8–13, 2022, Dublin, Ireland
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9201-3/22/07.
https://doi.org/10.1145/3502718.3524786

hours. In this section, we briefly summarize studies on the benefits of tutors [7, 9, 28, 32] and then describe how our work relates to the work conducted by Markel and Guo, the study that is most relevant to ours [19].

Prior work has examined the benefits of tutors [7, 9, 28, 32], but little has looked at the content of the student-tutor interactions themselves. Mirza et al. recently conducted a literature review of research on undergraduate tutors in computing to understand the benefits of undergraduate tutor (UT) programs and how to design these programs [23]. Mirza et al. summarizes the benefits of having tutors in computing programs: some of the major benefits of UT programs are that students perform better in courses [2, 5, 8, 9, 11, 26, 27, 34], are more motivated [8, 28], have higher satisfaction [3, 4, 7, 14, 22], and become better at communicating their thoughts and ideas [7, 9, 28, 32].

It has also been found that UT programs help tutors. This includes both social and cognitive benefits (e.g., improved connections with faculty [8, 12], improved appreciation and motivation for CS [13, 24]), technical skills (improved CS knowledge [3, 8, 9, 13], increased performance in their own coursework [13]), overall satisfaction [24, 29], and teaching skills [4].

The prior study most relevant to this study is the work by Markel and Guo, which has started to look at the content of student-tutor interactions through the metacognitive self-analysis of one undergraduate tutor [19]. The study summarizes the workflow and challenges (reading a student's mind, emotional regulation, maintaining student engagement, teaching vs. bug fixing, triage and prioritization, preserving student code, and receiving real-time feedback) that tutors face for a CS2 course [19]. Although this study begins to look at the content of the interaction itself, it is coming from the experience of a single tutor, and no work has looked at the interaction itself from an outside perspective.

Our study provides insight into the nature of student-tutor interactions based on actual footage of the interactions. Our work extends the prior work of Markel and Guo by providing a broader lens, as our footage comes from interactions between multiple tutors and students in two separate courses.

3 STUDY DESIGN

Our goal with this work was to gain an understanding of what happens during student-tutor interactions and whether these interactions are effective or ineffective. To do this, we asked the following research question: **What types and frequencies of behaviors are observed during student-tutor interactions?**

3.1 Course Context

We collected data during Fall 2020 in two lower-division undergraduate courses at our institution, a large, public, research-intensive institution in North America. Each course meets for 10 weeks followed by a week of final exams. The first course is an introductory programming course (CS1) taught in Java. The second is a computer organization and systems programming course (CompOrg) that is taught in C and 32-bit ARM.

Table 1: The number of students and tutors in the courses and the number who consented to being a part of the study.

	N Consented / Total	
Course	**Students**	**Tutors**
CS1	444 / 646	28 / 37
CompOrg	223 / 432	8 /12

3.2 Population

Our population of students and tutors comes from the CS1 and CompOrg courses. Students in the CS1 course are typically in their first year at the university, and CS1 is the first or second course a student takes in the CS department. Students in the CompOrg course are typically in their first or second year in the department, and CompOrg is the fourth course in the programming sequence. In order for a student to be a participant in this study, they must have consented to participating, they must have attended tutoring hours (a time period where students can get individual help from tutors), and the tutors from whom they received help must have also consented to participating in the study and uploaded the video for analysis.

In order to tutor for a course at our institution, the tutor has to have already passed the course and must take a tutor training course during their first quarter as a tutor. Our tutor training course includes one hour of instruction and five hours of work per week. Tutors learn how to communicate effectively with students and how to learn from and support others. This course walks tutors through examples of good and bad ways of interacting with students.

Tutors are often rehired to tutor for other courses: the tutors in this study had previously tutored in the CS department an average of 6.6 times for CS1 and 5.3 times for CompOrg . The average GPA of the tutors for both courses was 3.8. Note that this includes all tutors, not just those that participated in the study.

3.3 Data Collection

Students could request individual help from tutors during the designated tutoring hours that were available in each of the two courses. During these sessions, students could ask for help with tasks such as debugging their code and to clear up confusions about the material covered in lecture. All instruction, including these tutoring sessions, was held online via Zoom in Fall 2020 as an institutional requirement in response to the COVID-19 pandemic.

All tutors in our study were required to hold tutoring hours each week where they were available to help students with their programming assignments and general course questions. The number of students and tutors in each course and those who consented to participating in the study can be seen in Table 1. Only recordings where both the student and tutor consented to being a part of the research project were included in this study per our approved human subjects protocol.

3.4 Qualitative Analysis

Students seek help in tutoring hours for multiple reasons, the main two being (1) to get help debugging their programming assignments, and (2) to better understand the concepts and material from lectures. We chose to focus on videos in which students came for help with debugging their programming assignments, as these are common.

Analysis was performed by the first three authors of the paper. The first two authors began the open coding phase of the analysis to identify patterns in the interactions. There were seven rounds of open coding where they made sure to watch a wide variety of videos (e.g., multiple videos with the same tutor or same student). The result of these rounds was a list of codes that we believe accurately captures the behaviors we wished to analyze further.

Once we identified our coding scheme, we worked together to assign codes to parts of the interactions. We iteratively improved our codes by independently coding videos, followed by comparing our coding to identify errors in reliability. After two rounds of comparison, our third author began coding with no background knowledge aside from the code descriptions to test the reliability of our codes on someone who had not been a part of their creation. This round was used to determine the reliability of our ratings. All three authors watched the same six videos. Our inter-rater reliability for this round was 87.9%.

Having established reliability among the coders, we conducted the coding used in this study. We had a total of 718 videos of student-tutor interactions for CS1 and 184 videos for CompOrg. As we were unable to analyze all the videos due to the time required, we chose to pick a subset of videos for the analysis. We suspected that there might be variance in interactions based on the different tutors and students so we elected to randomly sample three videos per tutor. If a video was not about debugging (e.g., getting help with the course concepts), we skipped the video and randomly selected another. This occurred 18 times (21.4% of the time) for CS1 and 4 times (16.7% of the time) for CompOrg. For CS1, our analysis included 28 tutors which would result in a total of 84 videos; however two tutors only had two videos on debugging available, so the total number was 82. For CompOrg this included 8 tutors (24 videos).

3.5 Quantitative Analysis

Once we completed coding the videos, we examined the frequency at which each code appeared in the student-tutor interactions. We will report the quantitative analysis as the number and percentage of interactions of the videos analyzed.

4 RESULTS

Our process consisted of three separate steps conducted in different phases of the analysis: (1) identifying the codes, (2) collecting data on the frequency at which those codes appeared, and (3) understanding why these results may be meaningful to the community, including how the code may relate to effective (or ineffective) tutoring practices. To help with the readability of this section, we have organized our results by code, such that we will describe the code itself, how frequently it occurred, and implications of that finding.

4.1 Reason for attending tutoring hours

We coded the students' reasons for attending tutoring hours as either *C1.a. Compile error*, *C1.b. Runtime error*, *C1.c. Logic error*, when running the code produces the wrong result, or *C1.d. Other.* In the case of "other", the coders wrote out the reason for the interaction. For inter-rater reliability, the exact reasons were compared. Some such examples of the "other" category include, but are not limited to: trying to understand the syntax of the programming language or having issues running their program on their computer. Students may have sought help for multiple reasons.

We found that for CS1, the most common reason students attended tutoring hours was to get help with a logic error in their code (53.7% of the time). This was followed by having compile errors (25.6% of the time). In the CompOrg course, logic errors were also the most common issue for students to come for help (54.2% of the time). Interestingly, no students in the CompOrg course sought help with compile errors, but this is likely due to CompOrg being a later course in the introductory sequence where students already have a fair amount of experience addressing compiler errors.

The results can be seen in Table 2. We see that in CS1, students come to tutoring hours for runtime errors 6.0% of the time whereas in CompOrg this happens 29.2% of the time. This may simply be due to the large number of students attending for compile errors in CS1, meaning they didn't get to the point of having any runtime errors, but we also suspect that students in CompOrg needed more help with runtime errors because of challenges with memory management in C (e.g., segmentation faults).

Table 2: Reasons students attended tutoring hours.

	CS1		**CompOrg**	
Reason for attending	N	%	N	%
C1.a. Compile Error	21	25.6%	0	0.0%
C1.b. Runtime Error	5	6.0%	7	29.2%
C1.c. Logic Error	44	53.7%	13	54.2%
C1.d. Other	16	19.5%	3	12.5%

Table 3: Whether students explained their code and problem and whether the tutor asked them to.

	CS1		**CompOrg**	
	N	%	N	%
Student explains their *problem*				
Student *did not* explain	49	61.0%	13	54.2%
C2.a. Student explained	32	39.0%	11	45.8%
C2.b. Tutor asked student	4	4.9%	4	16.7%
Student explains their *code*				
Student *did not* explain	46	57.3%	8	33.3%
C3.a. Student explained	35	42.7%	16	66.7%
C3.b. Tutor asked student	5	6.1%	3	12.5%
Student explains both	18	22.0%	9	37.5%
Student explains neither	32	39.0%	6	25.0%

4.2 Students Engagement

We coded whether or not students explained their reason for attending tutoring hours (*C2.a. Student explained their problem*) and if, at any point, they explained what they think their code is doing (*C3.a. Student explained their code*). We also coded whether or not the tutor prompted the student to do either of those two things if the student did not do it on their own (*C2.b. Tutor asked student to explain their problem*, *C3.b. Tutor asked student to explain their code*). The results can be seen in Table 3. If a tutor asked a student to explain their problem or code, that means the student also explained their problem or code, so both codes are used (C2.a/b or C3.a/b).

We are concerned to find that in more than half of the interactions in CS1 (61.0%) and in CompOrg (54.2%), students neither explained their problem nor did the tutor ask them to explain their problem. Similarly, there were a number of students who never offered, nor were asked, to explain their code. Each of these elements seem important for a successful tutoring session, as tutors are taught that they need to engage with the student to understand where the student's difficulties might lie. By having students explain their problems and code, tutors can better identify any misconceptions held by the student to help take corrective action. Without that engagement with the students, tutors may be making assumptions about what the students do and do not understand.

Table 4: Reasons for compiling and running the students' code during the interaction.

	CS1		**CompOrg**	
	N	%	N	%
Code was compiled and run	44	53.7%	12	50.0%
C4.a. To see the error occur	10	12.2%	3	12.5%
C4.b. To see the output	12	14.6%	4	16.7%
C4.c. To debug	3	3.7%	2	8.3%
C4.d. To see if it works	30	36.6%	6	25.0%

4.3 Compiling and running the code

We coded whether or not code was compiled and run during the interaction and why it was run. This included *C4.a. To see the error occur*, *C4.b. To show the output*, *C4.c. To debug* (e.g., using gdb or print statements), and *C4.d. To see if it works.* The results can be seen in Table 4. Similarly, we coded whether or not any error messages or output were present (at all) during the interaction (*C4.e. Looked at error message or output during the interaction*). In this case, the code may not have been compiled and run during the interaction itself, but the student and tutor looked at error messages or output that had already been created by the student.

As these interactions are focused on helping students fix their programming assignments and debugging them, we would hope to see the students' code being compiled and run in every interaction. We found that in approximately half of the cases for both courses, no code was compiled and run during the interaction (see Table 4). This is concerning as all of these student-tutor interactions were focused on helping students debug their code. It is surprising and unfortunate to see that the code is run to debug in only 3.7% of the interactions for CS1 and only 8.3% in CompOrg.

Overall, we see that error messages or output were looked at (C4.e.) in 62.2% of the cases for CS1 and 50.0% of the cases for CompOrg. There are seven cases for CS1 and one for CompOrg in which error messages or output were looked at during the interaction but no code was compiled and run. This means the error messages or output was generated before the interaction. This is important as sometimes the student has not run their code since their last change, and it can lead to looking at outdated information and to trying to solve a problem that no longer exists.

4.4 Debugging

We coded whether or not the tutor helped the student debug during the interaction (*C5.a. Tutor helped Student Debug*). This includes the tutor asking a student to add print statements or tracing through the code, and running code to see the output. We separately coded whether or not the tutor said anything about the process of debugging during the interaction (*C5.b. Tutor mentioned debugging*). We assigned this code only when the tutor explicitly talked about the process of debugging. It is possible for a tutor to have helped a student debug without mentioning debugging (C5.a but not C5.b). For example, if the tutor said "let's try to figure out what is wrong with your program. To do that, let's put in a print statement..." then we did not assign C5.b. The word debugging is not used, but the tutor was clearly teaching the student the process of debugging. The results can be seen in Table 5.

We see that debugging is not mentioned and the tutor does not help the student actively debug their code in 79.3% of the cases for CS1 and in 79.2% of the cases for CompOrg. This suggests that debugging is not occurring during student-tutor interactions that should be focused on debugging. In many interactions, tutors debug students' code without including students in the debugging process, or they simply give the student the solution to their problem (e.g., the tutor looks over the student's code and finds the bug without talking to the student). We do however recognize that in some cases, a fix may be easy or a tutor may walk the student through an example and then end the interaction by telling the student to try to fix their problem (and debug) on their own, so it is not inherently bad for an interaction to have debugging neither mentioned nor performed, although we would expect to see it in the majority of interactions. This is a problem, as previous work has identified that lower-performing students are often unable to resolve their debugging issues [18]. Novice debuggers must apply many new skills simultaneously, such as understanding the intended operation of the program versus the execution of the actual program, general programming expertise, and knowledge of bugs and debugging methods [10]. However, novices' ability to use these skills is, at best, fragile, causing many to find it difficult and thus need help [25].

Table 5: Whether or not debugging was mentioned or performed during the student-tutor interaction.

	CS1		**CompOrg**	
Debugging	N	%	N	%
C5.a. Tutor helped student debug	13	15.9%	1	4.2%
C5.b. Tutor mentioned debugging	9	11.0%	4	16.7%
Both	5	6.1%	0	0.0%
Neither	65	79.3%	19	79.2%

4.5 Asking questions

We coded whether or not a tutor asked the student any questions during the interaction and what types of questions were asked. The types of questions included *C6.a. Clarifying Questions*, such as asking students to rephrase something they said or asking something about the code such as "Did you say that you tried x y z?" or "Is that an equals sign there or a minus sign?," *C6.b. Guiding Questions*, questions that are aimed at guiding students to the answer such as "What should be the return type for this method? And what type is your method currently returning?" or "Can you tell me what you know about the == operator? How have we used it in the past?" *C6.c. Asking questions to test whether the student understood the tutor's explanation*, and *C6.d. Asking a student if they understand*, potentially ineffective questions such as "x means y, right?" or "You understand that right?" The results can be seen in Table 6.

We find that in approximately one fifth of the interactions for both courses, the tutor does not ask the student a single question. We find similar percentages of the types of questions asked for each of the categories except for questions checking students' understanding where we see this happening in only 5% of the interactions for CS1 and in 25% of the interactions for CompOrg.

These are concerning numbers as without a dialogue, it is difficult to see how the tutor is engaging the student or adapting the tutoring to the student's understanding. Prior studies have shown the importance of tutors asking guiding questions during tutoring sessions [6, 17].

Table 6: Whether or not questions were asked during the interaction and what types of questions were asked, if any.

	CS1		CompOrg	
Types of Questions	N	%	N	%
Asked question(s)	68	82.9%	20	83.3%
No questions were asked	14	17.1%	4	16.7%
C6.a. Guiding	26	31.7%	9	37.5%
C6.b. Clarifying	53	64.6%	15	62.5%
C6.c. Checking understanding	4	4.9%	6	25.0%
C6.d. Ask if understands - yes/no	25	30.5%	10	41.7%

Table 7: The number of interactions where students left with the solution to their problem and whether the student figured out the solution or the tutor told them.

	CS1		CompOrg	
	N	%	N	%
C7. Left with the answer	67	81.7%	20	83.3%
C7.a. Student figured out solution	20	24.4%	5	20.8%
C7.b. Tutor gave the solution	53	64.6%	15	62.5%

4.6 Leaving with the Solution

We coded whether or not the student left with the solution to their problem (*C7. Student leaves with the solution*). If the student left with the solution, we kept track of whether the student figured out the solution to their problem (*C7.a. Student figured out the solution*), or the tutor explicitly gave the student the solution to their problem (*C7.b. Tutor gave student the solution*), (e.g., "Change the condition in the for loop from i <= arr.length to i < arr.length and that should fix it."). The results can be seen in Table 7.

In 82.9% of the interactions for CS1 , students left with the solution to their problem, and in 83.3% of the interactions for CompOrg. Leaving with the solution to their problem is not inherently bad or good. The tutoring hours are meant to help students get unstuck and allow them to continue moving forward on their assignments, but they are not necessarily meant to give students the solution to their problem. Unfortunately we see that in over 60% of the interactions for both courses, the tutor explicitly gives the student the solution to their problem. This means the student did not figure out the solution themself and was not guided to the solution but told the solution. This (C7.b) even occurs in cases where the tutor guides the student to the solution and then explicitly gives them the answer at the end rather than asking them to figure it out.

In a study on the differences in study habits of lower and higher-performing students, Liao et al. found that once lower-performing students get their code to work, they often move on without understanding why. The study also found concerning behaviors such as students not being able to explain their code at all, having assembled their code from bits and pieces they got from the course staff and friends [18]. As such, it's important to assist and guide students through debugging but not simply give away the answer.

4.7 Examples

We coded whether the tutor walked the student through any examples during the interaction (*C8. Tutor attempts to draw a picture or trace through an example*). This could be anything from drawing a memory diagram to tracing through the program with the student. This happened in 19 interactions (23.2%) for CS1 and 14 interactions (58.3%) for CompOrg. We hypothesize the larger number of examples in the CompOrg course is due to the large number of students that come to tutoring hours in CS1 for compile errors. There is not much to draw from a missing bracket, but for a runtime or logic error, there is more room for an explanation, although we do not believe this accounts for the near doubling in explanations that we see. Perhaps tutors in CS1 view the mistakes in the course as more basic and not requiring of diagrams, whereas they are more willing to provide diagrams in CompOrg for what they view as more challenging concepts (e.g., pointers, dynamic memory allocation).

4.8 Taking control of the students' screen

We coded whether or not the tutor took control of the student's screen during the interaction (*C9.a. Tutor takes control of screen*). The tutor might do this for multiple reasons, including to gain the ability to scroll through the code on their own or to highlight sections of code. We also kept track of whether or not the tutor typed in the student's code while they had control (*C9.b. Tutor types on student's screen*). In CS1, tutors took control of the students' screen in six interactions (7.3%) and typed on the students computer in four interactions (4.9%) . We saw no occurrences of this in CompOrg.

4.9 Commanding or interrupting the student

One code marked whether or not the tutor commanded (told) the student exactly what to type or what to do (C10. Tutor commands student). This does not include asking the student to add in print statements for debugging or pointing out a mistake a student made while typing. This code occurred in 13 interactions (15.9%) for CS1 and 1 (4.2%) for CompOrg . We did not expect this as tutors are taught not to do this during the tutor training course as it removes students from the decision process.

Another code recorded whether or not the tutor interrupted the student during the interaction (C11. Tutor interrupts the student). This does not include accidental interruptions such as accidentally speaking over one another. It included things such as interrupting a student saying, "No, no, no!" Interruptions happened in 20 interactions (24.4%) for CS1 and 4 interactions (16.7%) for CompOrg. We hoped to see no such interactions occurring, as previous work has found that students learn better with a polite tutor [33].

5 DISCUSSION

Our findings suggest that tutors are giving students the solutions to their problems and are often not teaching students the process of

finding and correcting their own errors. In this section, we discuss possible explanations for our findings and subsequent implications.
Tutor Training: The quality of tutor interactions was quite surprising given that our institution requires all first-time tutors to take a specific tutor training course. This course is intended to prepare and support first-time tutors in the computer science department and directly instructs tutors to do the opposite of what we commonly observed: we teach tutors to not give students answers but to help teach students the debugging process. In general, tutor training programs have been found to increase positive feedback from course instructors, increase tutor critical observations, and improve faculty-student interactions [12]. Our findings suggest that these programs, when implemented, may not be enough to support our tutors and train them properly.
Tutor Motivations and Challenges: We believe there are many factors that may lead tutors to give students solutions to problems rather than help them learn the process of solving their problems. A couple of factors we believe may be impacting these interactions include (1) tutors are also undergraduate students and therefore are tutoring their peers and, in some cases, their friends, which may make it more difficult to not give away the answer as they want to be liked by their peers, and (2) tutors often have many students waiting for help at once, in some cases over 60 students in the queue at a given time. This may lead them to rush through important steps in students' learning. Moreover, the study by Markel and Guo suggests that tutors face many challenges, some of which are even harder during online interactions, such as reading students' minds, emotional regulation, maintaining student engagement, teaching versus bug fixing, triage and prioritization, and preserving student code [19]. This is a daunting task for any teacher, but especially for an undergraduate who is tutoring their peers. Although our analysis did not have access to wait queue data for each session, future analysis could look to see if pressures from wait queues change tutor behavior.

5.1 Implications

Although students frequently leave these interactions with the solution to their present problems, we have to ask: what is the cost of not being taught to solve these problems on their own? We believe it may be giving students temporary relief but at the expense of possible long-term understanding. Recent research in computing has shown that students have not mastered the content instructors expect them to have learned in prior courses [16, 30, 31]. These student-tutor interactions may be a part of the reason we see students far along in the computing curriculum without the mastery of prior knowledge that we expect.

As mentioned in Section 4, previous work by Liao et al. that looked at the differences in behaviors in lower and higher-performing students found that once lower-performing students have the solution to their programming assignments, they move on without understanding why it works. Lower-performing students also often have code that is a mixture of code they got from the course instructional staff and their friends [18]. In our work, we did see cases of students starting their interactions with the tutors by saying things such as "The last tutor told me..." where they blame their error on the last student-tutor interaction while providing very little evidence (or none) on what they have tried on their own since working with the previous tutor.

When students are just given the solution to their problem and not taught how to debug on their own, we fear this may lead to an over-reliance on tutors and denies the student an opportunity to help correct erroneous mental models. This begs the question, do tutors have the correct mental models themselves? Do tutors understand that just giving students the solutions to their problems may not be effective? It would not be surprising if these tutors were also tutored similarly when they took the course and therefore believe that giving students the solution to their problem, without teaching them the process of finding and correcting their own errors, is a reasonable way to tutor. Future research on this topic should examine the reasons why tutors are behaving in this manner.

5.2 Limitations

Just as this is a unique situation that has allowed us to obtain video recordings of the student-tutor interactions, there are limitations to the study. One such limitation is that by recording the interaction, students and/or tutors may have acted differently due to the knowledge that they were being recorded. In addition, with all instruction online due to COVID-19, these interactions reflect the behaviors during a period of remote instruction and may not reflect these interactions when instruction is in person, and this is just examining student-tutor interactions at one institution.

Another limitation is that we were only able to include videos where both the tutor and student consented and where the tutor remembered to record and upload the videos. It is possible that tutors only uploaded videos that they may have deemed as good or acceptable. Although, if tutors were only selecting what they deemed were high-quality, that would imply that the tutoring sessions overall were less effective than what we observed.

6 CONCLUSION

In this study, we present an analysis of recorded student-tutor interactions where the student needed help debugging. The analysis is based on a series of codes that were developed using open-coding. The codes were designed to capture behaviors identified in the videos that may be reflective of effective (or ineffective) tutoring practices. We found that a concerningly large number of tutoring sessions result in the tutor giving away the answer (64.6% for CS1 and 62.5% for CompOrg) and an unfortunately small number of tutoring sessions teach the process of debugging (15.9% for CS1 and 4.2% for CompOrg). These students who seek out help are losing the opportunity to learn about the debugging process and to apply that process to find the present bug in their program. Moreover, such interactions may create a cycle of dependency where students become over-reliant on tutors to solve their debugging problems. Future work should examine reasons why these behaviors are common in tutors, the impact these behaviors have on student learning, and ways to improve the quality of student-tutor interactions.

7 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1650112 and NSF Award #2121592.

REFERENCES

[1] John R. Anderson, Albert T. Corbett, Kenneth R. Koedinger, and Ray Pelletier. 1995. Cognitive Tutors: Lessons Learned. *The Journal of the Learning Sciences* 4 (1995), 167–207.

[2] Maureen Biggers, Tuba Yilmaz, and Monica Sweat. 2009. Using collaborative, modified peer led team learning to improve student success and retention in intro cs. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09)*. 9–13.

[3] Saúl A. Blanco. 2018. Active learning in a discrete mathematics class. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. 828–833.

[4] Rebecca Brent, Jason Maners, Dianne Raubenheimer, and Amy Craig. 2007. Preparing undergraduates to teach computer applications to engineering freshmen. In *Proceedings of the 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports*. F1J–19.

[5] Mark J. Canup and Russell L. Shackelford. 1998. Using software to solve problems in large computing courses. *ACM SIGCSE Bulletin* 30, 1 (1998), 135–139.

[6] Michelene T.H. Chi. 2009. Active-constructive-interactive: A conceptual framework for differentiating learning activities. *Topics in cognitive science* 1, 1 (2009), 73–105.

[7] Adrienne Decker, Phil Ventura, and Christopher Egert. 2006. Through the looking glass: reflections on using undergraduate teaching assistants in CS1. In *Proceedings of the 37th ACM Technical Symposium on Computer Science Education (SIGCSE '06)*. 46–50.

[8] Paul E. Dickson. 2011. Using undergraduate teaching assistants in a small college environment. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. 75–80.

[9] Paul E. Dickson, Toby Dragon, and Adam Lee. 2017. Using undergraduate teaching assistants in small classes. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education (SIGCSE '17)*. 165–170.

[10] Mireille Ducasse and Anna-Maria Emde. 1988. A review of automated debugging systems: Knowledge, strategies and techniques. In *Proceedingsof the 10th International Conference on Software Engineering (ICSE '88)*. 162–171.

[11] Ronald Erdei, John A. Springer, and David M. Whittinghill. 2017. An impact comparison of two instructional scaffolding strategies employed in our programming laboratories: Employment of a supplemental teaching assistant versus employment of the pair programming methodology. In *Proceedings of the 47th IEEE Frontiers in Education Conference (FIE '17)*. 1–6.

[12] Francisco J. Estrada and Anya Tafliovich. 2017. Bridging the Gap Between Desired and Actual Qualifications of Teaching Assistants: An Experience Report. In *Proceedings of the 22nd ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. 134–139.

[13] Meg Fryling, MaryAnne Egan, Robin Y Flatland, Scott Vandenberg, and Sharon Small. 2018. Catch'em Early: Internship and Assistantship CS Mentoring Programs for Underclassmen. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. 658–663.

[14] TG Gill. 2005. Learning C++ "Submarine Style": A Case Study. *IEEE Transactions on Education* 48, 1 (2005), 150–156.

[15] K. Denise Kendall and Elisabeth E. Schussler. 2012. Does instructor type matter? Undergraduate student perception of graduate teaching assistants and professors. *CBE—Life Sciences Education* 11, 2 (2012), 187–199.

[16] Sophia Krause-Levy, Sander Valstar, Leo Porter, and William G. Griswold. 2020. Exploring the Link Between Prerequisites and Performance in Advanced Data Structures. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. 386–392.

[17] Mark R Lepper, Maria Woolverton, Donna L Mumme, and J Gurtner. 1993. Motivational techniques of expert human tutors: Lessons for the design of computer-based tutors. *Computers as cognitive tools* 1993 (1993), 75–105.

[18] Soohyun Nam Liao, Sander Valstar, Kevin Thai, Christine Alvarado, Daniel Zingaro, William G. Griswold, and Leo Porter. 2019. Behaviors of higher and lower performing students in CS1. In *Proceedings of the 24th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. 196–202.

[19] 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 (SIGCSE '21)*. 502–508.

[20] Jean Mckendree. 1990. Effective Feedback Content for Tutoring Complex Skills. *Human-computer Interaction* 5 (1990), 381–413.

[21] Douglas C. Merrill, Brian J. Reiser, Michael Ranney, and J. Gregory Trafton. 1992. Effective Tutoring Techniques: A Comparison of Human Tutors and Intelligent Tutoring Systems. *Journal of the Learning Sciences* 2, 3 (1992), 277–305.

[22] Mia Minnes, Christine Alvarado, and Leo Porter. 2018. Lightweight techniques to support students in large classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. 122–127.

[23] 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 15th ACM Conference on International Computing Education Research (ICER '19)*. 31–40.

[24] Elizabeth Patitsas. 2012. A Case Study of Environmental Factors Influencing Teaching Assistant Job Satisfaction. In *Proceedings of the 9th ACM Conference on International Computing Education Research (ICER '12)*. 11–16.

[25] David N. Perkins and Fay Martin. 1986. Fragile knowledge and neglected strategies in novice programmers. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*. 213–229.

[26] Inna Pivkina. 2016. Peer learning assistants in undergraduate computer science courses. In *Proceedings of the 46th IEEE Frontiers in Education Conference (FIE '16)*. 1–4.

[27] Stuart Reges, John McGrory, and Jeff Smith. 1988. The effective use of undergraduates to staff large introductory CS courses. *ACM SIGCSE Bulletin* 20, 1 (1988), 22–25.

[28] Eric Roberts, John Lilly, and Bryan Rollins. 1995. Using undergraduates as teaching assistants in introductory programming courses: An update on the Stanford experience. In *Proceedings of the 26th ACM Technical Symposium on Computer Science Education (SIGCSE '95)*. 48–52.

[29] Martin Ukrop, Valdemar Švábenský, and Jan Nehyba. 2019. Reflective Diary for Professional Development of Novice Teachers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. 1088–1094.

[30] Sander Valstar, William G. Griswold, and Leo Porter. 2019. The Relationship Between Prerequisite Proficiency and Student Performance in an Upper-Division Computing Course. In *Proceedings of the 50th Technical Symposium on Computer Science Education (SIGCSE '19)*. 794–800.

[31] Sander Valstar, Sophia Krause-Levy, Adrian Salguero, Leo Porter, and William G. Griswold. 2021. Proficiency in Basic Data Structures among Various Subpopulations of Students at Different Stages in a CS Program. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '21)*. 429–435.

[32] Andries van Dam. 2018. Reflections on an introductory CS course, CS15, at Brown University. *ACM Inroads* 9, 4 (2018), 58–62.

[33] Ning Wang, W. Lewis Johnson, Richard E. Mayer, Paola Rizzo, Erin Shaw, and Heather Collins. 2008. The politeness effect: Pedagogical agents and learning outcomes. *International journal of human-computer studies* 66, 2 (2008), 98–112.

[34] A.B. WeikleDee. 2016. More insights on a peer tutoring model for small schools with limited funding and resources. *Journal of Computing Sciences in Colleges* 31, 3 (2016), 101–109.