How Do Teaching Assistants Teach? Characterizing the Interactions Between Students and TAs in a Computer Science Course

Yana Malysheva
Washington University in St. Louis
St. Louis, MO
yana.m@wustl.edu

John Allen
Washington University in St. Louis
St. Louis, MO
johnjallen@wustl.edu

Caitlin Kelleher
Washington University in St. Louis
St. Louis, MO
ckelleher@wustl.edu

***Abstract*—Teaching assistants (TAs) play a crucial role in Computer Science courses. When a student is stuck or confused, they often rely on a TA to help them understand a concept or debug their program. At the same time, TAs in Computer Science courses are often very new at teaching, and somewhat new at programming. They may lack the knowledge and resources necessary to help students learn effectively. This work seeks to better understand the nature of TA-student interactions and identify potential opportunities for improvement. We conducted an observational study of one-on-one TA-Student interactions during office hours of a Computer Science course, and analyzed these interactions through the lens of known practices of effective one-on-one tutors. We found that TA-Student interactions focus on code over concepts, and this focus may be detrimental to TAs' use of good tutoring practices.**

I. Introduction

As enrollment in Computer Science (CS) programs grows[1], universities increasingly rely on undergraduate Teaching Assistants (TAs) to scale their classes effectively[2]. One of the main TA duties is providing one-on-one assistance to students during lab or office hours[2]. Students come to these office hour sessions when they are stuck or confused, and TAs take on the role of a tutor to help students work through through their issues and misconceptions.

Research on effective tutoring suggests that learning gains occur after a student reaches an impasse: they get stuck, encounter an error, or are uncertain about something [3], [4], [5]. Students come to TA office hours precisely because they are seeking help with this type of impasse. This means that office hour sessions hold a lot of potential for producing learning gains by helping students at the precise time when they are most receptive to learning. But to what extent is this potential fulfilled?

Little is known about the moment-to-moment interactions in TA office hours, and how these interactions affect student learning. We conducted an observational study of one-on-one TA-student interactions during office hours in order to understand whether TAs engage in effective tutoring practices, and whether there are specific barriers to good tutoring that can be addressed. Specifically, we sought to answer the following research questions:

RQ1. How does TAs' behavior during office hours compare with known effective tutoring practices?

RQ2. How does student behavior affect office hour dynamics and outcomes?

RQ3. Are there specific barriers to TAs using good tutoring practices?

II. Related work

Our work builds on two related areas of research: studies of undergraduate Computer Science TAs, and research on tutoring and what makes tutors effective.

A. Undergraduate CS TAs

Existing research on undergraduate TAs in Computer Science departments falls into two broad categories: evaluations of university TA programs, and studies of TA experiences.

1) Studies of TA programs: A recent systematic literature review [2] surveyed the prior work on Undergraduate TA (UTA) programs in computer science. They found that existing research describes several benefits of introducing UTA programs in Computer Science departments. Some studies identified improvements in students' grade performance [6], [7], [8] and attitudes towards the course [9], [10]. Studies also reported benefits to TAs, including increased confidence [11], [12] and improved interpersonal skills [13]. Finally, some papers described benefits to the instructors and department, including easing instructor workload [14], [15], [16] and building a general sense of community [15], [10], [16], [17].

Although these studies suggest that TA-student interactions can be beneficial to both TAs and students, they do not directly investigate the nature of these interactions. Our study seeks to analyze *how* undergraduate TAs interact with students during office hours, and how these interactions compare to tutoring practices used by effective tutors.

978-1-6654-4214-5/22/$31.00 ©2022 IEEE

2) Studies of TA experiences: Several recent studies seek to understand the TA experience - How TAs fulfill their responsibilities, what challenges they face, and what factors affect their behavior and satisfaction.

Riese et al.[18] analyzed reflection essays written by TAs and identified five categories of challenges TAs face, including the use of best practices and threats to best practices. Patitsas[19] used a combination of observations and semi-structured interviews with TAs to find six environmental factors that affect how TAs interact with students, and how they feel about their job as TAs.

Some studies focused specifically on what happens during TA office hours. Ren et al.[20] used the Design Recipe [21] methodology as a basis for designing short forms which TAs and students filled out before and after office hour sessions. These forms were used to both guide and track student help-seeking activity in office hours. Markel and Guo [22] presented an experience report of one TA conducting office hours, organized into a four-part model of TA-student interactions.

Several common themes emerge around factors that can present challenges to TAs, including: time pressures experienced when trying to help students[18], [19], [22]; the level of familiarity and comfort with the underlying material[22], [18]; and the tension between helping students learn and helping them achieve a better grade[22], [19].

These studies primarily focus on the TA's perspective, and often rely on TA reporting to capture what happens during TA-student interactions. In our study, we instead directly observe and analyze the interaction between TAs and students in order to understand how these interactions may affect student outcomes, and to identify potential opportunities for improvement.

B. Practices of effective tutors

Prior research on tutors and tutoring has sought to identify behaviors and practices that differentiate effective tutors from less effective ones.

In particular, studies have shown that effective tutors tend to provide indirect guidance rather than direct answers [23], [24], [25], [26], allowing students to come to the correct conclusion themselves [3], [4], [5], [27]. They often do this by asking questions that *lead* the student to an important idea or answer [24], [28], [25], [29], [30].

Effective tutors proactively seek to assess the student's current level of understanding and diagnose misconceptions [24], [25]. They also explicitly look for opportunities to generalize, summarize, and reflect on the work the student had just done, and connect it with the broader topics that they are covering [28], [24], [25], [3]

In this study, we seek to apply this research to analyze behaviors of teaching assistants conducting office hours. When helping students during office hours, TAs need to make similar decisions and apply similar pedagogical skills to the tutors described in the studies above. But in contrast to traditional tutors, TAs have less control over the tutoring scenario: their task is to help the student with the specific assignment and specific issue the student is asking about. So they cannot choose the topic of the tutoring session, guide the student's learning by assigning practice problems, or plan to reinforce concepts in follow-up sessions.

III. Methods

In order to understand the nature of TA-student interactions during office hours, we collected and analyzed recordings of office hours in an advanced Computer Science course held at Washington University in St. Louis.

A. Data Collection

We collected recordings of online TA-led office hours from one semester of an advanced-level undergraduate Programming Languages course. TAs for this class were selected from students who performed well in the class in prior semesters. All TAs are required to go through a mandatory training by the university, which focuses on appropriate interactions and anti-harassment training. There was no class-specific training. The TAs and the professor interacted regularly through a Slack channel, where the TAs were able to bring up common student issues and ask for help with problems they could not figure out themselves.

The TA office hours for the class were conducted remotely via Zoom, a teleconferencing application. All Zoom office hour sessions were recorded as part of the course. The course's professor then extracted and made available to the researchers those parts of the recordings where both the student and the TA had agreed to participate in this study.

Three TAs and 34 students agreed to participate in this study, out of a total of 4 TAs and 50 students in the class. 13 of these 34 students went to TA office hours at least once in the semester. Due to a technical problem, most of the office hours for one of the three TAs who consented to the study were not recorded. Altogether, we collected recordings from 26 office hour periods which represent 98 self-contained help sessions between a student and a TA. These recordings contained 8203 total utterances.

B. Data Preparation

The raw data consisted of 16 hours and 54 minutes of recorded TA-student dialog and automatically-generated transcripts. In order to code and analyze these recordings, we first had to correct the transcription and divide the recordings into self-contained problem-solving sessions where one TA was helping one student with a single problem.

The automatic transcription generated by Zoom was imperfect, especially when the participants used terms and phrases that are uncommon in day-to-day English conversations, such as specific functions, programming language constructs, and variable names. In addition, these transcriptions were sometimes wrong about who said a particular utterance, and other times merged several participants' overlapping utterances into a single sentence attributed to one of the speakers. To address these issues, two of the authors corrected and augmented the

generated transcripts to capture what was actually said by each participant.

In order to prepare the video recording data for coding by multiple raters, one of the authors split the data into individual problem-solving sessions, such that each session could be coded by a single rater without needing the context of other conversations outside of this one. The specific criterion for this split was that each session should cover a single conversation between the TA and one student, which addresses one specific self-contained part of an assignment that a student was having trouble solving on their own. Usually, though not always, this meant a single function that the homework asked them to implement.

C. Data Analysis

To understand how TAs and students interact during office hour sessions, and how this may affect the outcome of the sessions, we analyzed the data on two levels: per-session labeling and per-utterance coding.

1) Per-session Labels: In order to capture the nature of each problem-solving session, we labeled each session with three categories of metadata:

- Type(s) of issue addressed in this session (one or more type may apply to each session):
  - **Conceptual**: The student explicitly asked for help understanding and interpreting the concepts involved in solving the problem at hand.
  - **Implementing**: The session involved the TA and student working together to implement some part of a programming assignment.
  - **Debugging**: The session involved the TA and student working together to find and fix bugs in the student's code.
- Student's code state before coming to office hours: When the student came into this problem-solving session, did they have any code already written? Or were they asking for help with programming problems which they hadn't started working on yet?
- Outcome: Were the student and TA successful in resolving the student's issue(s) during this session?

2) Per-utterance Coding: We developed a coding scheme to categorize each utterance within a problem-solving session. This coding scheme seeks to describe the information flow between TA and student during the session, with a focus on the types of guidance the student asked for and the TA provided.

Starting with an initial candidate coding scheme, the authors iterated by independently coding a small subset of the data (two representative conversation samples totalling 82 utterances) and then coming together to discuss and resolve any disagreements and ambiguities in the scheme. The first author then updated the coding scheme based on the discussion, and the process was repeated. The authors went through this iteration process 6 times before converging on the final coding scheme.

The resulting coding scheme categorized each utterance along three dimensions:

- A label, chosen from 21 options shown in Table I, categorizing the purpose of the utterance in the context of the issue being solved
- A Boolean flag denoting whether this utterance shifted the conversation to a different sub-topic or context
- A Boolean flag indicating whether this utterance was phrased as a leading question rather than a direct statement.

The 21 per-utterance labels are split across four high-level types of utterances, based on the **information owner**: which participant knows (or is presumed to know) the information that's being discussed, and which participant is the intended recipient of the information?

- **Student to TA**: The intent is for the student to communicate something to the TA, usually about the issue they are having or the problem they are working on.
- **TA to student**: The intent is for the TA to communicate something to the student, usually some kind of guidance to help the student with the problem at hand.
- **Mutually created information**: Neither the student nor the TA have the information in question; the goal is to build a new understanding through the interaction. This most often happens during debugging: the student and TA are trying to understand and address the bug(s) in the student code.
- **Social glue**: Utterances that do not carry any information related to the problem at hand.

Note that these categories do not define the **speaker**. Either participant could make an utterance which belongs in any of these high-level categories. For example, a student requesting some specific type of guidance would be a "TA to Student" type of utterance.

Table I lists the full set of classification labels within these four high-level categories, and provides an example of each type of utterance. The guidance levels in the "TA to student" category are a particularly important subset of labels: they capture how explicitly and directly the TA guided the student toward an answer. Research suggests that effective tutors try to provide indirect answers which allow the tutee to make their own cognitive leaps[24], [25], so less explicit guidance levels may be indicative of greater adherence to good tutoring practices. A full document describing the detailed criteria for each category is available online [31].

We measured inter-rater reliability separately along each of these dimensions. To do this, each author independently coded the same randomly-chosen subset of the data (around 10% of the total data set). The Fleiss' kappa measure for the utterance label was 0.74, indicating substantial agreement. For the context change flag, the Fleiss' kappa was 0.82, indicating very good or "almost perfect" agreement. For the leading question flag, the Fleiss' kappa was 0.50, indicating moderate agreement. The inter-rater agreement was much lower for the leading question flag than the other two dimensions, in large part because there were very few leading questions asked. For example, in the subset of data used to determine inter-rater

Table I
UTTERANCE CLASSIFICATION LABELS

Type	Label	Example
Student to TA	Issue the student needs help with	Student: If you had time, I just would love some help debugging this insert function
	Establishing context	TA: Sorry, did you send what you have? Oh yeah, okay it's right there.
	Symptoms of the issue	Student: my test empty is passing in the brackets, and test full is failing.
	Student's approach so far	Student: I was thinking of like trying to delete each element... and count[ing] them...
	Student's knowledge state	TA: Um, so have you ever worked with the "E..." thing before?
TA to Student	(Guidance 5) Explicit answer	TA: So just put an 'a right before "treenode"
	(Guidance 4) Explicit algorithm or solution	TA: You want to create a local binding with the head and the tail of the list and then...
	(Guidance 3) Strategizing: step(s) to resolve the issue	TA: What I would recommend is putting it in a data structure that you can reverse easily
	(Guidance 2) Declarative description of state	TA: [Your function is] calling both insert and remove in the same calls.
	(Guidance 1) Building understanding	TA: So, the way the "above" [function] works is...
	(Guidance 0) General relevant resources	Student: What are the helper functions that would be helpful, like from the library?
	Generalizing, Reflecting, Making connections	TA: Yeah, usually with SML, the first thing is like "we should probably add a helper"
	Tangential information	Student: Is partial credit available?
Mutually created information	Reproducing the issue	TA: Let me pull up my Racket thing and see what happens when I try to run mine.
	Making changes and analyzing the result	Student: What if I add "val item = rootnode" ... "#2"
	Referencing TA's own solution	TA: Let me see if I can pull up how I did it... I had dictionary...
	Referencing third-party solutions	TA: Yeah, [the professor's solution] does it in quite an interesting way.
	Diagnosing student's misconception	TA: Can you explain why you structured it like that?
	Code (or system) comprehension	TA: I think what happens is... what happens? how does it make the matrix?
Social glue	One's internal state as it relates to the problem	Student: It's reassuring, I guess, to know that I have like the right structure, hopefully.
	Conversational social glue	TA: Hey, how's it going?

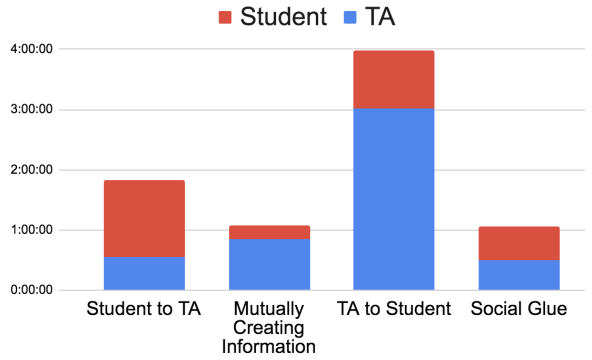

Figure 1. Distribution of total speaking time by category of utterance and by speaker role

reliability, between 1 and 5 utterances(depending on the rater) out of 808 were labeled as "leading questions".

IV. RESULTS

Figure 1 summarizes how speaking time was distributed across the four utterance categories and the two participant roles (TA and student). The majority of the time (50.1%) was spent on TA-to-student utterances, which is expected, since the main purpose of office hours is for TAs to provide guidance to students.

Notably, the participants did not spend much time in the Mutually Creating Information category, which is the category that contains all debugging utterances. In fact, they spent nearly as much time on Social Glue interactions as on Mutually Creating Information. This is in line with the more detailed findings described below: both TAs and students tended to avoid debugging behaviors in favor of rewriting code to match the TA's own solution to the problem.

In the subsections below, we address each of our research questions: (RQ1) How does **TA behavior** during office hours compare with known effective tutoring practices? (RQ2) How does **student behavior** affect the office hour dynamics and outcomes? (RQ3) Are there specific **barriers** to TAs using good tutoring practices?

A. TA Behavior

TA behavior deviated from known effective tutoring practices in several important ways: (1) TAs tended to guide students through their own solution to the problem, instead of adapting their guidance to the individual student's approach; (2) TAs tended to provide very direct and explicit guidance; and (3) TAs rarely made specific types of utterances that are likely to build student understanding

1) Guiding students through TA's own implementation: We found that when guiding students, the TAs in our study relied heavily on stepping students through their own solution to the homework problem. Each of the TAs in the study had taken the same course either the previous semester or the year before, and they retained the code they had written when they took the course.

Thus, it was easy and convenient for the TAs to pull up their own solution to the current problem and use it to guide the student. Notably, although there were a handful of cases where TAs tried to debug the student's existing code by comparing it to their own (93 total utterances across 35 distinct

sessions), most of the time the TA simply read off the detailed algorithm that their code implemented, regardless of whether this matched the student's approach, for example:

> TA: "...so then I used mlet, and for the list for mlet I passed in a list with _x and e1 and _y and e2, and then for the expression to my mlet I check..."

TAs tended to offer this type of guidance without trying to understand or address the student's existing approach to the problem. The participants did discuss the student's approach in 72 out of 98 sessions, but in 50 of those cases, the student was the first to bring up their approach. In one example, the student had to insist on explaining their approach twice before the TA realized it was a better way of doing things:

> Student: "Yeah, I was thinking because it's sorted, [explanation of student's algorithm]"
> TA: "Yeah, the way I approached is, [longer explanation of different algorithm]"
> Student: "Okay, I was thinking if you have like the list 4,5,6,7 and you're looking for five, [explanation of algorithm using concrete example]"
> TA: "Yeah, that's true. That is more effective."

After this exchange, the TA focused on helping the student interpret the error message, and together they were able to correct the issue while sticking with the student's original intent.

Interestingly, in the very next session with the same TA and the same student, a similar exchange happened when the student asked for help diagnosing an error message. The error was actually caused by a minor bug where the student did not update a function signature when they changed the function's behavior.

> Student: I'm getting this error that says [student describes the error message and hypothesizes where the problem may be]
> TA: Yeah, maybe I suggest just making that function a helper function and [TA suggests a way of completely restructuring the code to make it closer to the way TA solved the problem]

In this case, the student seemed less sure about their approach, and did not insist on explaining their rationale. So, the student spent the next 10-15 minutes rewriting their entire function under the direction of the TA. Because the student and the TA were thinking about the problem differently, and because the TA did not clearly explain the high-level overview of their approach, the process took several iterations and was longer than it needed to be. The student and TA never identified the original bug, instead completely changing the buggy line in the process of restructuring the code.

2) TAs provide direct and explicit guidance: As we described in the related work session, research on practices of effective tutors suggests that giving indirect guidance helps students learn better. But when TAs focused on describing their own solution to the student, they frequently did this in the form of very direct and explicit guidance.

In our coding scheme, the labels "Guidance 4: Explicit algorithm or solution" or "Guidance 5: Explicit answer" capture guidance which gives the student direct and explicit answers about the code they should be writing. TAs resorted to this kind of explicit guidance in about 75% of the sessions (74 out of 98).

The TA would sometimes give this type of explicit answer even when the student was asking a more general and indirect question, for example:

> Student: "In the unit test, it's- it's expecting it to be curried?"
> TA: "Exactly. So this is what you want it to look like. [Pastes code] It's a simple mistake."

TAs gave a response that was **more** explicit than the student's question in 11.5% of the cases. They gave a **less** explicit response in 7.5% of the cases. The rest of the time, when TAs responded to direct requests for guidance, the guidance they provided matched the student's requested level of guidance. However, only 23% of TAs' guidance utterances were in direct response to student request for guidance - the rest of the time, TAs offered explanations and guidance without the student making a specific request.

3) TAs rarely build understanding: In addition to categorizing direct guidance provided by TAs along an explicitness spectrum, our coding scheme captures several other types of utterances that could be indicative of good tutoring practices: asking leading questions, generalizing or reflecting, and diagnosing misconceptions. These types of utterances were extremely rare in the dataset. Out of the 8203 total utterances, TAs in the study asked leading questions around 16 times; tried to generalize or reflect on lessons learned 37 times; and attempted to diagnose student misconceptions 3 times.

Each of these types of utterances focus on student understanding of the underlying material, rather than the correctness of their current code. The fact that they are exceedingly rare indicates that TAs tended to focus on the goal of making the student's code work, rather than building an understanding of why it works.

B. Student Behavior

Students who come to office hours seem to be content with the TAs' strategies of focusing on code and providing explicit guidance which gets their homework to pass unit tests. Students often come into office hours expecting to be walked through a solution to a homework problem, and they seem to adapt their questions to individual TAs' guidance style, asking more explicit questions of TAs who tend to provide more explicit guidance.

1) Students expect to be walked through an implementation: In 49 out of 98 sessions, student directly asked for and received help implementing part of their homework assignment. And in 39 out of 98 sessions, students came in looking for help on a problem without having written any code beforehand. In these situations, students often did not have any specific questions beyond "I would like your help writing code for this problem".

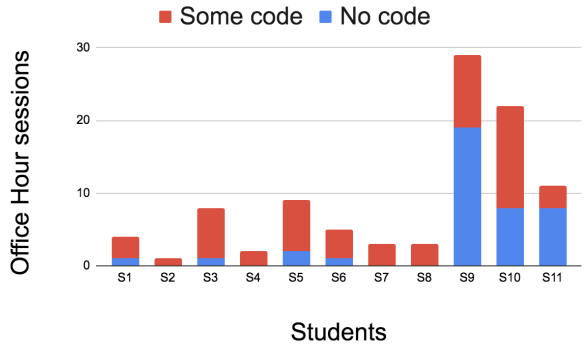

Figure 2. Instances where students came into office hours with code written vs. with no code written

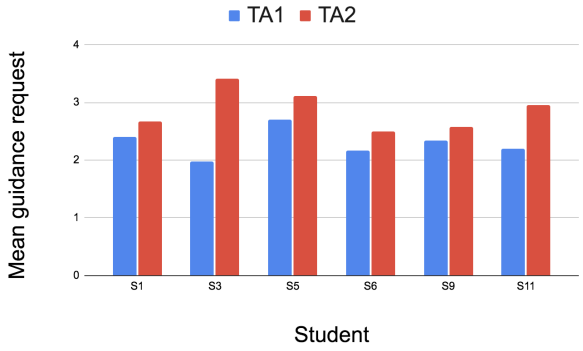

Figure 3. Explicitness level of questions asked of two different TAs by the same set of students

TAs tried to probe the student about their approach to the problem in 22 of 98 total sessions. But it was sometimes difficult to get the student to engage:

> TA: "So for brackets, what are you thinking so far? Like what's your approach?"
> Student: "To be honest, I have no clue."
> TA: "So what - do you know what brackets is trying to do, like, what the goal of the method is?"
> Student: "Not clearly, no."

This kind of interaction may further discourage TAs from taking the initiative and asking students to articulate their approach or thought process.

At the same time, having the experience of being walked through a solution step-by-step in past office hour sessions may encourage students to ask for help before attempting to solve the problem themselves. Figure 2 shows how many times each student came into office hours with and without code written. We can see that students who came to office hours regularly were much more likely to come in without any code prepared than those who only came in occasionally.

Given that students come in with the apparent expectation that the TA will guide them through the entire implementation process, and they don't easily engage in talking conceptually about their approach to the problem, it can be hard for TAs to go against those expectations and keep the conversation focused on building conceptual understanding rather than creating code which passes the unit tests.

2) Students ask more explicit questions of TAs who provide more explicit guidance: Both of the TAs for whom we have full data, TA1 and TA2, fit the behavior patterns described in the previous section. But TA2 consistently provided more explicit guidance, and was more likely to provide answers that were more explicit than the question they were answering.

As we can see from Figure 3, students seemed to adapt to individual TAs' guidance level: For each of the six students who attended office hours of both TAs, the same student asked more explicit questions of TA2 than they did of TA1. So, the choices that the TA makes when providing guidance can not only have an effect on the outcome of the immediate problem the student is facing, but also set the tone for future interactions between that student and that TA.

C. Barriers to using good tutoring practices

We observed several patterns which could be indicative of barriers to TAs using good tutoring practices: (1) The difficulty of the class may have limited TAs' comfort level with the content and their ability to adapt their guidance to individual student needs; (2) both TAs and students tended to view the focus of office hours as helping students fix their code, rather than helping them learn; and (3) the office hour sessions tended to be conducted under a lot of time pressure.

1) The difficulty of the content makes less effective tutoring practices more enticing: The Programming Languages class we observed for this study introduces many unfamiliar paradigms and programming languages at a rapid pace. Because of this, even experienced students and TAs can find the class more challenging than they are used to.

In 26 out of 98 sessions in our dataset, the TA was unable to resolve the issue that the student needed help with. This means that TAs were able to help the student about 73% of the time.

However, that success rate improved dramatically when the student came in **without any code prepared** - 35 out of 39 such sessions resulted in successfully resolving the student's issues. Similarly, the success rate was quite high for sessions where the student explicitly asked for help **implementing** part of their homework - 44 out of 49 such sessions had a successful outcome. Together, these two types of sessions have a success rate of 87.5% (49/56). In both of these types of sessions, the TA could usually resolve the issue by simply walking the student through the TA's old solution to the relevant problem.

By contrast, when students came in with already-written code and wanted help with debugging or conceptualizing, the TAs were only able to help the student in 57% (24/42) of the cases.

For example, in several instances, the TA thought the logic of the student's code looked right, and could not understand why the tests weren't passing:

> TA: Yeah, logically, like map looks good. That's always the problem. It looks good. It doesn't work.

Other times, even falling back on the strategy of comparing the student's code to the TA's own solution did not help them find the issue:

> TA: It's so weird that it doesn't work. I mean, it looks exactly like mine. If you want to post on Piazza and maybe [the professor] has some clue as to why it's not working.
> Student: Yeah, I feel like it might be just some like crazy, weird like quirk or something.
> TA: Yeah. SML can be like that sometimes.

TAs also sometimes struggled with providing conceptual explanations when students asked questions about their own code. For example, one student came into office hours with already-working code, but wanted to understand how and where a specific operation happened within an intricate sequence of currying and recursion. The student and TA discussed the code for about 10 minutes. Although the TA seemed confident about their understanding of the code, they were unable to pinpoint the specific operation the student was looking for or articulate why it's actually working the way it's working. Eventually, the student gave up and decided to refer back to the lecture videos:

> Student: "So, since f1 doesn't take any parameters, how is that possible?.."
> TA: "Let me- I have some notes..."
> Student: "If it's hard to find I can, like, rewatch the videos."
> TA: "Yeah, I think in the videos they explain it. Also in the hints and gotchas. I think that might give you- I'll post the link"

Given these kinds of difficulties, it makes sense that both students and TAs gravitated toward strategies like asking for help before getting started on the implementation, and providing such help by stepping through an existing working solution. These strategies feel safer and more rewarding because they make a successful outcome more likely.

2) Perception of the goal of office hours: Both TAs and students seemed to view the goal of office hours as helping the student submit a correct solution to the homework problem, rather than ensuring that the student has the conceptual understanding necessary to solve the problem.

As described above, students often came in with the expectation of being walked through an implementation of a particular function or programming problem. And TAs tended to conform to those expectations by stepping through their own solution from previous semesters.

Most student questions focused on code over concepts: out of the 98 help sessions in our study, only 10 were about purely conceptual questions. In addition, in 17 sessions, some conceptual questions came up in the process of resolving other issues.

When students did ask for help in understanding concepts, TAs often redirected the student to reviewing class material or talking to the professor. This may be because TAs did not view providing conceptual guidance as part of their responsibilities.

For example, in one session, a student tried to get a more conceptual understanding of what was going on after the TA walked them through creating a working implementation:

> Student: "I'm really struggling with, like, what it means for like... returning functions and this whole lambda stuff."
> TA: "Yeah, kind of like currying in Racket?"
> Student: "Yeah. Not really clicking right now, but hopefully it will."
> TA: " If you want to ask [the professor] to meet, then he's pretty good at explaining it."

3) Time pressure: Helping students in real time puts pressure on the TA to understand and resolve the issues quickly. Moreover, in the office hour sessions in this study, there were frequently multiple students present in the Zoom call waiting their turn while the TA helped the current student. These pressures likely increased TAs' cognitive load, and incentivized them to take the most expedient, simplest path to addressing the student's issues. Students also seemed to feel this time pressure: many unresolved sessions ended when the student yielded their time to allow the TA to help someone else or because the TA's office hours were close to being over, for example: "I don't wanna hold you for too long" or "Oh, your hours are so done. Um, alright, well, I will just like go another TA's hours later"

The median session duration was 8 minutes 5 seconds, and the average duration was 10 minutes and 20 seconds. But debugging is a very time-consuming process, and trying to debug complex code in a few minutes can be daunting.

In addition, a lot of this time was taken up by interactions meant to convey factual information about the student's code state to the TA: 11.4% of total on-topic(non-"Social Glue") speaking time was taken up by utterances **establishing context** (e.g. what function the student is working on, whether the code is pushed to the repository) and describing **symptoms of the issue** (e.g. reading or pasting the error message the student saw). This accounted for 43% of all speaking time in the "Student to TA" category of utterances. Spending this much time on logistics took away from the time and attention that the participants could have spent on analyzing the actual underlying issues.

V. Limitations

This is a small qualitative study limited to the behavior of three TAs and 11 students in one advanced Programming Languages class.

Because of the small scale of the study, the findings might not generalize to be broadly applicable to Computer Science teaching assistants at large.

In particular, it is possible that some of the TA-student dynamics we observed are specific to advanced classes, and that TA-student relationships work differently in more introductory courses. For example, some prior research noted that students in introductory classes can be hard to engage in a discussion, and instead "are often passively listening and nodding"[22]. But in our observations, students were actively involved in

the conversation, and generally seemed to have a friendly, peer-like relationship with the TAs: TAs and students had comparable amounts of speaking time, and the participants felt free to spend a large portion of the time on off-topic (social glue) interactions.

VI. Future Work

There are two major avenues for future research: more thorough investigations of TA-student interactions, and design and analysis of potential interventions to address the barriers that make it harder for TAs to use good tutoring practices.

A. Investigating TA-student interactions

Larger-scale studies are needed to investigate whether our qualitative findings can be replicated, and whether they generalize to other contexts, such as introductory courses and courses at different institutions.

Additionally, future work could investigate how various factors affect student-TA interactions. Do external factors such as the number of students seeking help, or the difficulty of the current problem, affect TA behavior? How does TA behavior affect student behavior and expectations?

Finally, our study did not attempt to directly measure how TA-student interactions affected student outcomes in the class. In particular, future work could investigate how a student's solution to a particular problem changes after they get help with it in office hours; and how coming to TA office hours affects students' ability to solve similar problems in the future.

B. Designing interventions to address barriers

Our study suggests that there are several barriers that make it harder for TAs to engage in effective tutoring during office hours: the difficulty of the underlying concepts and tasks; the TAs' expectations about the goal or purpose of office hours; and the additional difficulty imposed by the time pressure of trying to understand the issue and debug and write code in real time, with people watching.

These barriers could potentially be alleviated by tools and interventions that assist TAs in their tasks. For example, tools that automatically analyze student code and present actionable information to the TA could both alleviate the difficulty of debugging and save time spent on establishing context. And interventions that scaffold the flow of the TA-student interactions may help establish and reinforce better expectations about the goal of office hours.

VII. Conclusion

In this study, we found that TAs do not always follow good tutoring practices when assisting students during office hours. Instead, they offer overly-explicit guidance which doesn't adapt to the student's approach, and focus on ensuring the student's code works over building the student's understanding of underlying concepts. Students are content with this approach, and often come to office hours seeking help without first attempting the problem.

We identified three specific barriers that may make it harder for TAs to use good tutoring practices. These barriers may prevent students from thoroughly understanding the material that is taught in the class. In particular, the TAs' focus on correctness of submitted code over comprehension may artificially inflate some students' grades in the course without ensuring they understand course material. We recommended a direction for future work that could address these barriers, and therefore may help significantly improve student learning in these classes.

Although this is a small-scale qualitative study conducted in a single advanced CS course, it is a first step in understanding how moment-to-moment interactions between TAs and students affect student learning.

References

[1] Stuart Zweben and Betsy Bizot. 2017 CRA Taulbee survey. *Computing Research News*, 30(5):1–47, 2018.

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

[3] Kurt VanLehn, Stephanie Siler, Charles Murray, and William B Baggett. What Makes a Tutorial Event Effective? page 6, 1998. 39.

[4] Kurt VanLehn, Stephanie Siler, Charles Murray, Takashi Yamauchi, and William B. Baggett. Why do only some events cause learning during human tutoring? *Cognition and Instruction*, 21(3):209–249, 2003. 475.

[5] Caroline P. Rosé, Dumisizwe Bhembe, Stephanie Siler, Ramesh Srivastava, and Kurt VanLehn. The role of why questions in effective human tutoring. In *Proceedings of the 11th International Conference on AI in Education*, pages 55–62, 2003.

[6] Maureen Biggers, Tuba Yilmaz, and Monica Sweat. 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*, pages 9–13, 2009.

[7] Ronald Erdei, John A. Springer, and David M. Whittinghill. 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 *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–6, October 2017.

[8] Inna Pivkina. Peer learning assistants in undergraduate computer science courses. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–4. IEEE, 2016.

[9] Adrienne Decker, Phil Ventura, and Christopher Egert. Through the looking glass: reflections on using undergraduate teaching assistants in CS1. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 46–50, 2006.

[10] Mia Minnes, Christine Alvarado, and Leo Porter. Lightweight Techniques to Support Students in Large Classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 122–127, Baltimore Maryland USA, February 2018. ACM.

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

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

[13] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Jaakko Kurhila. Massive increase in eager TAs: Experiences from extreme apprenticeship-based CS1. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 123–128, 2013.

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

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

[16] Andries van Dam. Reflections on an introductory CS course, CS15, at Brown University. *ACM Inroads*, 9(4):58–62, 2018. Publisher: ACM New York, NY, USA.

[17] Stuart Reges. Using undergraduates as teaching assistants at a state university. *ACM SIGCSE Bulletin*, 35(1):103–107, January 2003.

[18] Emma Riese, Madeleine Lorås, Martin Ukrop, and Tomáš Effenberger. 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*, pages 547–553. Association for Computing Machinery, New York, NY, USA, June 2021.

[19] Elizabeth Patitsas. A case study of environmental factors influencing teaching assistant job satisfaction. In *Proceedings of the ninth annual international conference on International computing education research*, pages 11–16, 2012.

[20] Yanyan Ren, Shriram Krishnamurthi, and Kathi Fisler. What Help Do Students Seek in TA Office Hours? In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 41–49, 2019.

[21] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to design programs: an introduction to programming and computing*. MIT Press, 2018.

[22] Julia M. Markel and Philip J. Guo. 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*, pages 502–508, Virtual Event USA, March 2021. ACM.

[23] Mark R. Lepper, Michael F. Drake, and Teresa O'Donnell-Johnson. Scaffolding techniques of expert human tutors. 1997. 268.

[24] Mark R. Lepper and Maria Woolverton. The wisdom of practice: Lessons learned from the study of highly effective tutors. In *Improving academic achievement*, pages 135–158. Elsevier, 2002. 210.

[25] Xin Lu, Barbara Di Eugenio, Trina C. Kershaw, Stellan Ohlsson, and Andrew Corrigan-Halpern. Expert vs. non-expert tutoring: Dialogue moves, interaction patterns and multi-utterance turns. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 456–467. Springer, 2007. 27.

[26] Xin Lu, Barbara Di Eugenio, Trina C Kershaw, Stellan Ohlsson, and Andrew Corrigan-Halpern. Tutorial Dialogue Patterns: Expert vs. Non-expert Tutors. page 8, 2006. 4.

[27] Jennifer G. Cromley. What Do Reading Tutors Do? A Naturalistic Study of More and Less Experienced Tutors in Reading. *Discourse Processes*, 40(2):83–113, September 2005.

[28] Willem S. De Grave, Diana HJM Dolmans, and Cees PM Van Der Vleuten. Profiles of effective tutors in problem-based learning: scaffolding student learning. *Medical education*, 33(12):901–906, 1999. 255.

[29] Michael Glass, Jung Hee Kim, Martha W. Evens, Joel A. Michael, and Allen A. Rovick. Novice vs. expert tutors: A comparison of style. In *MAICS-99, Proceedings of the Tenth Midwest AI and Cognitive Science Conference*, pages 43–49, 1999. 51.

[30] Hyeun Me Chae, Jung Hee Kim, and Michael Glass. Effective Behaviors in a Comparison Between Novice and Expert Algebra Tutors. page 7, 2005. 18.

[31] Yana Malysheva. Classification of interactions during office hours. https://tinyurl.com/TAStudentCoding.