Interrelation between Teaching Assistants' debugging strategies and adherence to sound tutoring practices during office hours

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

Caitlin Kelleher
Washington University
USA
ckelleher@cse.wustl.edu

Barbara Jane Ericson
School of Information, University of Michigan
USA
barbarer@umich.edu

Abstract

Office hours often play an important role in Computer Science courses. But office hours are often conducted primarily by graduate and undergraduate Teaching Assistants (TAs). These TAs may have limited experience with both debugging issues that students may encounter in their code, and with utilizing good tutoring practices when guiding the student toward a solution. In order to understand TAs' challenges and strategies during office hours, we conducted an observational study of TAs holding office hours for a Data Oriented programming course. We found that TAs tend to use fewer good tutoring practices in help sessions where they've encountered more difficulty with debugging the student code. We also identified three dimensions of debugging strategies where the TA's choices may either exacerbate or alleviate the difficulty they have with both debugging and guiding the student. We believe that these insights can help inform the design of tools and interventions to help TAs conduct more effective office hours.

CCS Concepts

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

ACM Reference Format:
Yana Malysheva, Caitlin Kelleher, and Barbara Jane Ericson. 2024. Interrelation between Teaching Assistants' debugging strategies and adherence to sound tutoring practices during office hours. In *24th Koli Calling International Conference on Computing Education Research (Koli Calling '24), November 12–17, 2024, Koli, Finland*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3699538.3699562

1 Introduction

Office hours can play an important role in Computer Science (CS) courses, since they are one of the few opportunities for students to get one-on-one help with the course material. Office hours in CS courses are often conducted at least in part by Teaching Assistants (TAs), who are usually either undergraduate or graduate students with some experience in the course material. Since the TAs are themselves students, their experience in both tutoring and debugging can vary widely. Therefore, the extent to which a particular office hour session is helpful to the student can also vary.

In CS courses, students usually come into office hours seeking help with a specific programming problem or task. The TA's goal is to both help the student resolve the specific issue they are having with their program, and help the student learn some skill or concept that may help them resolve similar problems on their own in the future. Accordingly, we examine two aspects of conducting office hours where TAs may experience varying degrees of difficulty:

(1) Understanding and **debugging** what is happening in the student's program
(2) Using sound tutoring practices to **guide** the student toward understanding something they previously didn't.

We refer to these aspects as "debugging" and "guiding" throughout the rest of the paper.

Previous qualitative research has suggested that TAs do not always use sound tutoring practices to guide the student during office hours [12, 23], and that the difficulty of finding and fixing bugs in student code under time pressure could be a contributing factor to this problem [23]. In this work, we conduct a deeper quantitative and qualitative analysis of the connection between a TA's difficulties debugging the student's code during a particular office hour session, and the difficulty they have guiding the student during that same session. Specifically, we pose two research questions:

- **RQ1**: What is the relationship between the difficulty a TA has **debugging** and the difficulty they have **guiding** the student in a particular help session?
- **RQ2**: What **debugging strategies** do TAs employ that may affect their ability to effectively **guide** the student?

To answer these research questions, we conducted an observational study of TA-student interactions during office hours across two semesters of a Data Oriented Programming course. We found that there is a positive correlation between how much difficulty the TA had with debugging student code, and how much difficulty they had in utilizing known good tutoring practices for guiding the students. We also identified three specific dimensions where a TA's choice of strategy for debugging the student code may affect the difficulty that TA has in guiding the student.

2 Related Work

2.1 TA-student interactions in Computer Science courses

Mirza et al. [26] conducted a systematic literature review of the prior work on Undergraduate TAs (UTAs) in CS. The studies described

Koli Calling '24, November 12–17, 2024, Koli, Finland

ACM ISBN 979-8-4007-1038-4/24/11
https://doi.org/10.1145/3699538.3699562

in this literature review suggest that introducing TA programs can benefit both TAs [2, 8, 35] and students [1, 5, 6, 25, 29]. However, these studies were primarily concerned with the organization of TA programs in CS departments, and did not directly touch on the interactions between TAs and students.

In recent years, several studies have explored TA-student interactions from the perspectives of both TAs and students. Lim et al. [19] found that students' desired outcome of office hours did not always match the students' own expectations of what is beneficial for their learning. Likewise, several studies of TA experiences [24, 28, 30] identified a tension between the TAs' goals of helping a student learn and helping them achieve a better grade by resolving their current issues. Two recent observational studies [12, 23] show that TAs often have difficulties following sound tutoring practices during office hours. Goletti et al. [9] and Cheng et al.[3] investigate interventions which give TAs explicit guidelines for using good tutoring practices.

While several prior works describe the challenges TAs face in both debugging student code and using good tutoring practices, our work is focused on the **relationship** between these two common areas of difficulty. In particular, we investigate how a TA's approach to debugging the student code may impact their adherence to good tutoring practices.

2.2 Practices of effective tutors

Prior research on effective tutoring techniques identifies several patterns of behaviors that may lead to more effective tutoring. In particular, effective tutors prefer to provide indirect guidance instead of directly giving away the answer [17, 18, 21, 22]. They often ask leading questions to help the student focus their thought process [4, 31, 33, 34]. This can give students the opportunity to arrive at the important conclusion themselves, and thus retain a better understanding of the concepts involved [4, 31, 33, 34].

In our study, we use these patterns to inform our metrics of effective tutoring during office hours. Specifically, we measure TAs' use of explicit vs. indirect guidance, and their use of leading questions when helping the student.

2.3 Novice debugging

Since Teaching Assistants are still students themselves, they often lack extensive experience when it comes to debugging code, especially code that was written by someone else. When debugging student code during office hours, they are likely to utilize common novice debugging strategies and experience pitfalls that are typical of novice debuggers. Research on novice debugging behaviors has uncovered several behavior patterns that are relevant to our work:

Novices often utilize pattern-matching and tinkering techniques to make changes to the code without understanding the code or the effect of the change. For example, they may change code that looks "suspicious" simply because it's different from what they are used to seeing, or more generally make trial-and-error changes based on their intuition of what *might* help [7, 20, 27, 36]. When novices do try to understand what is going wrong, they often prefer using forward reasoning — tracing through the code — over backward reasoning — trying to understand the output and reasoning about how it may have been caused by the code [7, 10, 36].

Finally, several studies have shown that self-explaining - explicitly verbalizing explanations of one's thought process and understanding of the problem - is linked to better performance and outcomes on debugging tasks [16, 20, 32].

3 Methods

In order to understand how TAs' debugging strategies affect office hour dynamics and outcomes, we collected and analyzed recordings of office hours for an intermediate-level course on Data-Oriented Programming. The office hours we collected were conducted over videoconferencing by both undergraduate and graduate teaching assistants.

3.1 Data Collection

We collected recordings of office hours during two semesters of an undergraduate course on Data-Oriented Programming. Students in this course are expected to already be familiar with basic programming concepts. The course is focused on teaching the students how to accomplish data-oriented tasks in Python, such as using APIs, scraping HTML, and extracting and aggregating data. This course employs both undergraduate and graduate teaching assistants. Since both graduate and undergraduate students had the same responsibilities during office hours, for the purpose of this study, we did not differentiate between graduate and undergraduate teaching assistants.

In this course, some office hour sessions were conducted in person, and some were conducted over Zoom, a videoconferencing application. We only recorded and analyzed office hours that were conducted over Zoom, as recording all of the relevant information during in-person office hours would be prohibitively complicated and intrusive.

7 TAs and 170 students agreed to participate in this study. We asked the TAs who were participating in the study to use Zoom to record and upload the videos of their help sessions in cases when the student they were helping was also participating in the study.

In practice, not all TAs who agreed to participate in the study recorded and uploaded the eligible help sessions they took part in. We collected and analyzed a total of 12 help session recordings, which together comprised 167 minutes of video data. 4 different TAs and 8 different students took part in these help sessions. This study design was reviewed and approved by the university's Institutional Review Board (IRB).

3.2 Data Preparation

To prepare the data for analysis, we first used an automatic transcription service (Otter.ai) to create first-draft transcriptions of each of the help sessions. Since the automatic transcription service is geared toward transcribing everyday language, the first-draft transcriptions contained a lot of errors when the conversation revolved around programming jargon and the specific code being discussed. Therefore, two researchers manually corrected the generated transcriptions.

To capture data about the TA and students' problem-solving strategies, one researcher who was familiar with the content of the course also transcribed two sets of data about the problems being addressed during each help session:

(1) **The focus of the discussion**: Instances when the participants are directly discussing something on the student's screen, such as a particular line of code or the current state of the program's output. For each instance, the researcher recorded what specific line or item was being discussed.
(2) **Edits made**: Instances when the student made edits to their code. For each instance, the researcher recorded:
   - Whether the TA directly suggested making this specific edit. For example, "What happens if you get rid of 'results' on line 201?" or "I would suggest using that json.loads function".
   - Whether this edit was intended to correct a problem, or was a diagnostic edit, such as printing out a value or temporarily commenting out broken code to see output from a different line of code.
   - In cases where the edit was intended to correct a problem, whether it was successful (the code moved toward being more correct) or not (the edit made the code more wrong and/or did not address the existing problems).

3.3 Data Analysis

In order to analyze TA-student interactions during office hour help sessions, we coded the utterances in the help sessions using a coding scheme based on the scheme developed in [23]. This coding scheme categorizes each utterance into one of four high-level categories, and one of several possible sub-categories in each category. The four high-level categories capture the **intended direction of information flow** for that utterance: who has (or is presumed to have) the information being discussed, and who is receiving this information?

- **Student to TA** utterances — the information in question is something the student knows, such as what issue they want help with, or how they were approaching the problem.
- **TA to student** utterances — the information in question is something that the TA knows (or is assumed to know), such as how the student should change their code to fix a bug.
- **Mutually Creating Information** (MCI) utterances — nobody currently knows or has the information in question. The student and TA are trying to gain an understanding that neither of them has. These are almost always **debugging** utterances - trying to build an understanding of what is happening and how to fix it.
- **Social Glue** utterances — utterances which do not carry any information related to the problem.

A particularly important subset of the "TA to student" high-level category is the set of six guidance levels (Guidance Level 0 through 5). These sub-categories capture how directly and explicitly the TA was guiding the student to the answer. Research suggests that indirect guidance can lead to more effective learning than explicitly giving away the answer [18, 21], so this set of categories is an important measure of how the TA is adhering to good tutoring practices.

Table 1 shows the full list of categories we used, with an example for each category. We merged three pairs of sub-categories from the original coding scheme [23], because we found that it was often hard to draw a distinction between them, and the distinctions did not provide meaningful information in the context of our research questions. For example, we merged "Referencing TA's own solution" and "Referencing third-party solutions", since the source of the solution being referenced did not matter in our context.

In addition to these categories, the coding scheme includes two boolean flags that indicate important events:

- Whether this utterance is a **leading question** — a question to which the speaker already knows the answer, but they think asking it will lead the other person to a useful realization.
- Whether this utterance represents a **context change** — a change in the specific topic of the discussion. For example, the student asking a new unrelated question; or the TA deciding to give up on the current approach and try something different.

We measured inter-rater reliability separately for the main classification scheme and the two boolean flags. Two researchers independently coded around 20% of the data, and we measured the inter-rater reliability using Fleiss' kappa. For the main classification categories, Fleiss' kappa was 0.69, indicating substantial agreement. For the context change data, Fleiss' kappa was 0.84, indicating very good agreement. For the leading question data, Fleiss' kappa was 0.52, indicating moderate agreement. The leading question data was fairly sparse, since the TAs in this data set did not ask leading questions very often. This may have contributed to the lower Fleiss' kappa score.

3.4 Data Filtering

Since the main focus of our work is TA debugging behavior, we were specifically interested in sessions where the TA did at least some debugging. For this reason, we excluded those sessions where either (1) the TA made no utterances in the "Mutually Creating Information" category — since this is the category that contains debugging utterances, the TA must not have needed to debug anything during those sessions; or (2) there were no edits made to code during the entire session — if the session involved any debugging, we would expect the TA and student to touch the code in some way.

Based on these criteria, we excluded 3 out of 12 sessions from our analysis. These three sessions primarily dealt with clarification questions about problem statements, grading criteria, and validating a student's high-level approach.

4 Results

We used both quantitative and qualitative analysis of the data to answer the two research questions we posed:

(1) **RQ1**: What is the relationship between how much difficulty a TA has **debugging** during office hours and how much difficulty they have using good tutoring practices to **guide** the student?
(2) **RQ2**: What **debugging strategies** do TAs use during office hours which may increase or decrease the amount of difficulty the TA encounters in guiding the student?

Table 1: Classification Categories

Type	Label	Example
Student to TA	Issue or symptom	Student: "So I ran it and I don't know why it's giving me this message."
	Establishing Context	TA: "Okay, let me take a look at your find_bio_names function."
	Student's approach so far	Student: "I was trying to get the birth years using regex"
	Student's knowledge state	TA: "Do you remember how you can convert a variable from one type to another?"
Mutually created information	Reproducing the issue	Student: "Here, let me run it again."
	Code (or system) comprehension	TA: "So we're getting an error, but it's somewhere else now."
	Proposing or trying changes	TA: "And then let's just run and see what happens."
	Referencing External Resources	TA: "I'm going to double check it next to the other one"
	Diagnosing misconceptions	TA: "Why did you decide to do ".load"?"
TA to Student	(Guidance 5) Explicit answer	TA: "So, actually, it's hand.read"
	(Guidance 4) Explicit algorithm	TA: "Then we're gonna want to write that variable to the file"
	(Guidance 3) Step(s) to resolve the issue	TA: "The way I would build that, is I would build it a very small amount at a time."
	(Guidance 2) Declarative description of state	TA: "that tells me that there's not a key called "results" when you go through it."
	(Guidance 1) Building understanding	TA: "So, you know how with turtles, you would name the turtle something?"
	(Guidance 0) General relevant resources	TA: "Shift+tab is how you go backwards."
	Generalizing, reflecting, making connections	TA: "We love new errors, new errors are the best."
	Social Glue	Student: "Oh, well, that went fast."

4.1 RQ1: Relationship between difficulty debugging and difficulty guiding

To analyze the relationship between the difficulty a TA has with debugging and understanding the student's problem and the difficulty they have in using good tutoring practices to guide the student, we first performed Bayesian Estimation to establish whether there is a significant relationship between measures of these two dimensions of difficulty.

In recent years, researchers have argued that Bayesian methods are better suited than traditional Null Hypothesis Significance Testing (NHST) for several areas of research on human behavior, including Psychology [14], Human-Computer Interaction [11], and Organizational Sciences [15]. In particular, researchers argue that Bayesian analysis provides data that is more informative, easier to interpret, and more suitable for small-n studies [11, 14, 15].

We found that there is, indeed, a credible positive correlation: in a given help session, the more difficulty a TA has in debugging issues, the more difficulty they are likely to have in using good tutoring practices to guide the student. To gain a deeper understanding of this relationship, we then qualitatively compared two help sessions that were conducted by the same TA, but were on the opposite ends of the spectrum when it came to each dimension of difficulty.

4.1.1 Correlation between difficulty debugging and difficulty guiding. We used two metrics for each of the two dimensions of difficulty, which we derived from the coded utterance data.

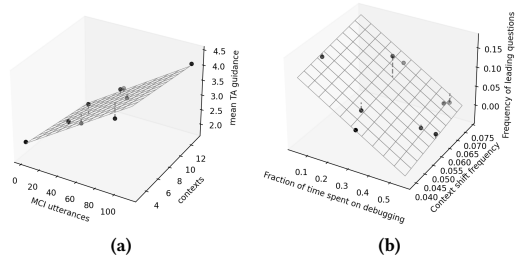
For difficulty debugging, we measured:

(1) The number of **Mutually Creating Information** (MCI) utterances in the help session — this number captures **how much** debugging the TA and student had to do in order to resolve the student's issue.
(2) The number of **Context Changes** that happened during the help session — these often indicate that the student and TA encountered an additional complication: "go ahead and make sure you open up your- oh wait, why do you have "file not found"?" or may have abandoned their current line of inquiry because they felt they exhausted it: "Okay, anyways, we tried going through that other rabbit hole...". These types of shifts both indicate that the participants are encountering difficulties, and may themselves cause increased cognitive load.

For difficulty in using good tutoring practices to guide the student, we measured:

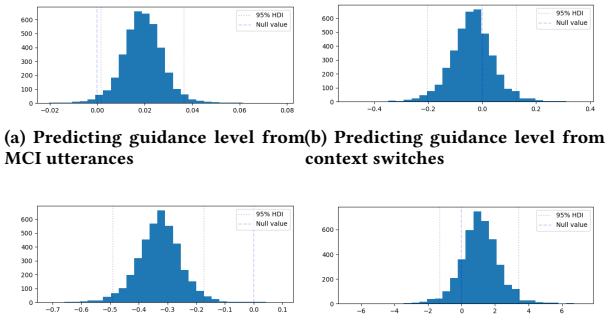
(1) The average **Guidance Level** the TA used when providing guidance to the student — as described in the Methods section above, more explicit guidance (higher guidance level) means that the TA is giving away the answer to the student, which may not be conducive to effective learning [18, 21].
(2) The number of **Leading questions** the TA asked during the help session — phrasing information in the form of a leading question encourages the student to engage with the question and draw their own conclusion.

We used Bayesian Multiple Linear Regression (BMLR), a Bayesian Estimation approach to multiple linear regression, to assess how well the **difficulty debugging** metrics together can predict each of the **difficulty guiding** metrics.



(a) (b)

Figure 1: Regression results for predicting guidance level (a) and leading questions asked (b) using metrics of difficulty debugging.



(a) Predicting guidance level from MCI utterances

(b) Predicting guidance level from context switches

(c) Predicting leading questions from MCI utterances

(d) Predicting leading questions from context switches

Figure 2: Histograms of credible values for the slope parameters of predicting difficulty guiding using metrics of difficulty debugging.

In Bayesian Estimation, a hierarchical model is first chosen to describe the relationship between several variables. We used a robust multiple linear regression model [13, 15]. This model describes a linear relationship between two or more independent variables (in our case, the metrics for "difficulty debugging") and one dependent variable (in our case, we performed this analysis separately for each of the metrics for "difficulty guiding"). In addition, the noise in the data is modeled as having a t-distribution around this linear model. A prior belief distribution is then chosen for each parameter in the model. We used the mildly-informed priors described in Chapter 18 of Kruschke(2014) [13].

Given the model, the prior distribution, and our set of data, we can now estimate the posterior distribution for the model parameters. To do this, we used Stan[1], a popular software package for performing Bayesian analysis.

To decide whether we can use the posterior distribution to draw credible conclusions about the relationship between the variables, we analyze how the distribution of certain parameters relates to some null value. In the case of linear regression, we can analyze how the posterior distribution of the slope for each independent variable relates to the slope of 0, which would indicate no relationship between the variables. We compute the 95% Highest Density Interval(HDI) - the highest-density interval which covers 95% of the probability distribution. If the null value lies outside of that interval, we conclude that this distribution represents a credible relationship between the variables.

Our first BMLR model uses the number of MCI (debugging) utterances and the number of context changes to predict the average guidance level used by the TA. For this model, we used the total number of debugging utterances and context changes as our independent variable metrics. These totals depend in part on the total length of the help session, in addition to the moment-to-moment difficulty the TA may have had debugging. This total session length is, in itself, a salient and important component of the overall complexity of the session, and is important to capture in this case.

Our second BMLR model uses the *fraction* of time spent on MCI (debugging) utterances and the *frequency* of context changes (measured in utterances) to predict the frequency of leading questions asked by the TA during the help session. For this model, we used fractional metrics of the independent variables instead of the totals we used in the first model, because we anticipated that using total values for all metrics would introduce a confounding factor. All three of the metrics involved (debugging utterances, number of context shifts, and number of leading questions) would correlate positively with the total length of the debugging session. However, we anticipated (and observed) a predominantly negative numeric correlation between the measures of difficulty of debugging and the measure of difficulty guiding: We expected the metric for leading questions asked to decrease, but the metrics for difficulty debugging to increase, when the TA has more difficulty with both aspects of conducting office hours. In this case, eliminating this confounding factor was more important than capturing the aspect of difficulty that comes from the total length of the help session.

Table 2 shows the estimated posterior mean and standard deviation for each of the parameters in the two models. Figure 1 shows the regression planes of each model plotted against a scatter plot of the data. Figure 2 shows the estimated posterior distribution of each of the slope parameters in the two models. From the histograms of credible values, we can see that the metric associated with **debugging time** does have a credible linear correlation with each of the metrics of difficulty guiding the student. In both models, the 95% HDI area does not intersect with the null value. However, for the metric associated with **context changes**, the null value does fall squarely within the 95% HDI in both models. Therefore, at least in the context of these multiple-regression models, we cannot conclude that the number or frequency of context changes has predictive power over the difficulty the TA will have with guiding the student. This may be because the context shift variable is simply redundant in the multiple regression model, and the variable for debugging time already captures all of the predictive power of the model. The context shift metric may also be too noisy to be predictive. Nevertheless, this analysis shows a credible correlation between the difficulty a TA has debugging code during a help session and the difficulty they have in using good tutoring practices during the same help session.

[1]https://mc-stan.org/

Table 2: Posterior means and standard deviations of parameters in the BMLR model predicting metrics of difficulty guiding (guidance level and number of leading questions asked) from metrics of difficulty debugging

	Guidance Level		Leading Questions	
Parameter	Mean	Std. Dev.	Mean	Std. Dev.
Z-intercept	2.28	0.47	0.1	0.07
Slope (debugging utterances)	0.02	0.0088	-0.33	0.08
Slope (context changes)	-0.04	0.08	1.1	1.21
σ of t-distribution	0.4	0.17	0.03	0.01
ν of t-distribution	33.16	30.11	30.36	28.53

4.1.2 Comparison of two contrasting sessions. The statistical analysis described above suggests that at least one measure of the difficulty the TA had debugging in a help session predicts how much difficulty the TA had using good practices when guiding the student. However, this prediction does not necessarily imply a causal relationship. It could be true that encountering difficulties in the debugging process makes the TA less able to tutor effectively. But it could also be true that some other latent aspect(s) of a help session affect both types of difficulty. To gain insight into why both types of difficulties occur in some sessions but not others, we compare two contrasting sessions conducted by the same TA. We found that the TA's approach to debugging was similar across the two sessions, and reflected common novice debugging practices. This approach proved effective in the first, easier session; but in the second session, the approach led the TA to pitfalls which hindered both effective debugging and effective guiding.

The first session occurred during the fifth week of class (out of 15). The student was looking for help with a homework assignment which introduced the concept of multiple classes interacting with each other. Most of the student's issues were based in conceptual misunderstandings — the student was confusing classes and objects, and was having trouble understanding when to use the keyword "self". This session went very well with respect to the metrics of difficulty guiding. Out of all sessions in our dataset, it had the highest number of leading questions asked (8) and lowest value for the mean level of guidance the TA provided (2.19). The majority of the guidance the TA provided was phrased as guidance level 2 (declarative description of state), e.g.: "something we want to keep straight is "self". So whatever class self is in, that's what self refers to". The session also went reasonably well with respect to the metrics of difficulty debugging. There were 8 total context changes in the session, which was the median number across our dataset. And 28% of the session was spent on debugging utterances, which is slightly less than the median (31%). This session took around 14 minutes, which is less than the medians session length of about 16 minutes. After this session, the student had a more correct version of their code and a demonstrated ability to find and fix similar errors on their own.

The second session occurred during the twelfth week of the same semester. The student in this session was working on a homework assignment which concerned querying an API, caching data in a local file, parsing the JSON that the API returned, and aggregating values. The student had a mostly-working solution and needed help finding and fixing a few small bugs. This session had the worst values in the dataset for each of the four metrics of difficulty. 58% of the session was spent on debugging utterances, and there were 13 total context changes. The TA asked no leading questions. The mean level of guidance the TA provided was 4.1, because the majority of the guidance was phrased either as level 4 (explicit algorithm), e.g. "And then we're gonna want to write that variable to the file" or level 5 (explicit answer), e.g. "And then it's at request_url just like [line] 124". This session took nearly 34 minutes, which makes it the longest session in the dataset by far — the second longest session was around 18 minutes. The student and TA did eventually find and fix the two original bugs. But they also introduced a new subtle bug which remained unfixed when the session ended.

The overall approach. Although the outcome of these two sessions differed drastically, the TA's approach to debugging and guiding the student was quite similar across the two sessions. In both of these sessions, the TA's primary strategy for locating issues was to scan through some promising sub-section of the code and look for things that seemed problematic to the TA. The TA would make a decision about a change they wanted to make to the code, and then guide the student toward that change using varying degrees of explicitness. The TA did not make many attempts to involve the student in their problem-solving process in either session. Although the TA occasionally explained their thought process to the student, this was almost always after the fact, as a side note: "That's why I wanted to start with the other function to get you warmed up" (from the first session); "I just sort of wanted to clear it, but..." (from the second session).

This approach closely maps onto the sub-optimal novice debugging strategies that we discuss in section 2.3: The TA used forward reasoning to trace through code, then tinkered with parts of the code that look different from what the TA expected. They did not engage in self-explaining or verbalizing their thought process in the moment.

First help session. The TA was able to immediately identify several errors in the student code and begin explaining the problems to the student: "So there's already something that I've seen the misconception with a ton of students already. So it's totally normal." They explained when the student should be using a class name and when they should use an instance of the class, and led the student toward fixing one of the cases of using a class name instead of an instance.

The session continued with the TA pointing out and discussing other problems they noticed with the code. For each problem, the

TA would try to lead the student toward their intended solution indirectly, e.g. "we want to call the dining halls student list, but we don't actually have a dining hall to access". Sometimes the TA would phrase the guidance as a leading question, e.g. "So how come we were allowed to use dining_hall above? The lowercase version?".

In some instances, when the student was unable to use higher-level hints to arrive at a solution, the TA would increase the level of explicit guidance until they "bottomed out" and told the student what change they should make, e.g. "we don't need the self at the beginning"; "so where you have that Dining_Hall in capitals, should be in lowercase".

As the session went on, the student began applying the concepts that the TA had been explaining to find and fix similar errors without the TA's direct intervention: "Can I just put... Oh wait, self because it'll be in student class?"

Second help session. In the second help session, the TA had significantly more trouble finding the bugs in the student code. After the student explained that one of their functions was throwing and catching an exception, the TA started by scanning the code in that function from the top down: "So you have a request URL from get_request_url(list), that's good. And then you do read_json on the file name, so that's good. "if request URL is in datadict.keys", that's good."

When the TA reached a particular line of code which seemed suspicious, they paused and examined it silently for around 20 seconds. After that, the TA jumped to telling the student exactly what to change, without providing any justification: "see how in line 128 You're doing read_json? I would suggest using that json.loads function that we went over in class instead". In this case, the change the TA suggested was correct, because the read_json function was for reading json from a file, but the student was trying to parse json from a string. This also happened to be the bug that was causing the current exception, so when the student re-ran the code, it resulted in a new error message, and the TA was able to conclude that the fix worked. However, the TA didn't seem to connect the exception to the bug in any way.

The TA then spent nearly 20 minutes trying to locate the next bug. Similar to the first bug, the TA used the error message to identify a starting place for examining the code, and then looked for potential changes to make in that general area: "Okay, so we're going to 133 that is where the error is coming up first." The TA briefly acknowledged the contents of the error message — that a function call in the call stack expected a string, but got a dictionary — but did not seem to apply this information to inform their debugging. Instead, the TA made a series of specific suggestions for changes to the code in the vicinity of the error message. Before they were able to find the actual error, the TA made 5 separate suggestions for how to change the code. All of these were either wrong (making the code more incorrect) or unnecessary (not changing the behavior of the code), and the TA rarely explained the reasoning behind their suggestions. For example, the TA insisted on changing how the student's code reads and writes JSON from file:

> TA: Okay, so a lot of you are doing just "load" instead of "loads". Why did you decide to do ".load"?
> Student: [...] I thought that load was meant for [loading from] the file.
> TA: **Okay, yeah.** What I'm gonna suggest you do is we change the format for both.

Even though the TA seemed to agree that the student's approach was valid, they insisted on splitting up the functionality into two steps - reading the file and converting the string into JSON. This change did not affect the code's behavior, and the TA did not explain the purpose of this change.

In another attempt to fix the same bug, the TA did provide a speculative justification to a change they suggested: "Why don't we change that to say if new_dict["status"]="OK" as its value? Because maybe what's happening is the status isn't "OK". And there's actually some kind of error." However, this justification did not make much sense, since the if statement in question was working correctly in the test case they were trying to debug: the if condition passed, and the exception was being triggered by a line of code inside of the if block. After the student made the suggested edit, nothing immediately changed in the execution, because the altered if condition still passed. However, this did make the if condition less correct in general. The new condition throws an exception in a different test case, where the status key did not exist. This new failure mode went undetected until the very end of the help session. At that point, the TA and student could not identify what was triggering this new error message before the TA ended the help session.

Discussion. The TA's strategy of scanning sections of code to find potentially problematic areas worked much better in the first session than in the second one. In the first session, the TA was quickly able to identify parts of the code that were wrong, *and* immediately knew why they were wrong, based on the TA's understanding of the underlying concepts. However, in the second session, the problems in the student's code were more subtle and specific to the student's approach to the problem. Therefore, when the TA found sections of code that looked different from what they expected, these sections did not always correspond to actual bugs or problems. And when the TA attempted to debug these sections by tinkering with the code, the changes they made were frequently unproductive or outright wrong.

Although the TA did not verbalize their debugging thought process in either session, this was not as noticeable in the first session, when the TA did not have to do much debugging to identify the problems. But in the second session, the TA would frequently fall silent for long stretches of time or simply express their confusion about the code: "This function is like very convoluted...", "Give me just... give me one second because there's a lot going on in this function".

The TA's strategies of forward reasoning, tinkering, and not verbalizing their thought process acted as effective debugging shortcuts when the problems in the student code were relatively easy, and did not hinder their ability to guide the student. In the first session, the TA was able to quickly jump to the correct conclusions, and then demonstrated good use of sound tutoring practices such as using leading questions and giving indirect guidance by describing what was happening in the student code and why. But when the problems got harder, these same strategies seemed to undermine the TA's ability to both debug and guide effectively. In the second session, those same strategies did not allow the TA to develop a

clear understanding of what was going wrong with the student code, and consequently made it harder to provide good explanations to the student. Thus, the TA primarily resorted to telling the student what to change.

4.2 RQ2: TAs' debugging strategies which may have an impact on the TAs' difficulty in guiding the student

In order to understand how different TAs approach the debugging strategies we identified in section 4.1.2, and how this affects their success in both debugging and guiding, we conducted both a qualitative and quantitative analysis of all of the office hour sessions in our dataset. Specifically, we identified three dimensions of debugging strategies corresponding to the three suboptimal debugging practices we observed in the case study in section 4.1.2:

- Suggesting changes to student code: are the suggested changes a result of tinkering and pattern-matching, or are they well-reasoned changes?
- Finding and diagnosing bugs in student code: does the TA use forward reasoning to scan code, or backward reasoning to interpret the output and trace the bug backwards from it?
- Communicating their thought process: does the TA think aloud and explain their thought process to the student?

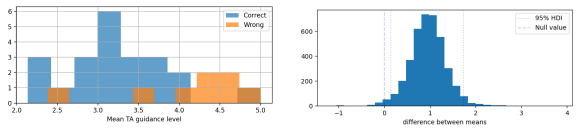
We first examined the sessions qualitatively to understand the range of different approaches TAs took for each dimension. We then used the coded utterance data to draw quantitative conclusions about the relationship between the range of debugging approaches and the TAs' ability to guide the student.

4.2.1 Suggesting code changes. As part of the debugging process, TAs sometimes chose to make a direct suggestion for an edit the student should make to their code. Some of these suggestions were well-reasoned and made sense in the context of the current issue. However, other times, the TA simply wasn't sure how to fix the current issue, and thus speculatively suggested changes that may or may not help: "go back to your folder for the homework, just go ahead and delete the cache file. If not, I have ideas for your other function but I thought maybe we would check this first."

These types of suggestions often turned out to be wrong, and the speculative change either made the program more incorrect, or did not address the current issue. In the example above, the cache file was not related to the issue the student was having. The TA thought that the deleted cache file would regenerate after they ran the code, but this was not the case. After deleting it, the TA and student spent a significant amount of time trying to recreate it. Other times, TAs introduced new bugs by suggesting changes that were wrong and were not reversed by the end of the help session.

This pattern of suggesting changes that may be wrong was most problematic when the TA did not explain or motivate the change. If no explanation is provided, the student does not have much information to reason about whether the change might make sense, and what side effects it might have. Moreover, the TA is also less likely to have thought the change all the way through than in cases where they talk through why they are suggesting a change.

By contrast, sometimes the TA suggested a change they weren't completely sure would work, but only after talking through what they think might be happening, and why this change might fix it. For example, one TA noted that the key "results" did not work when trying to index into a complex data object, but similar indexing on other lines worked correctly when "results" was omitted. They then suggested trying to delete that index: "I don't know if the whole thing is going to work but it seems like that results key was blowing up." This change did, in fact, turn out to be correct because the nested JSON structure did not have the "results" key that the incorrect code expected. Even though the TA did not dig through the entire object to verify that this was the case, their guess was both educated and well-motivated.



(a) Histograms of guidance levels that preceded correct and incorrect TA edit suggestions

(b) Histogram of posterior distribution of the difference between the means

Figure 3: Relationship between the correctness of TA-suggested edits and the guidance that the TA provided prior to suggesting the edit

To gauge the effect of talking through and motivating one's suggestions for code changes, we considered all edits that were made directly at a TA's suggestion, and were either correct (moved the code closer to being correct) or wrong (made the code less correct, or were not relevant to the issue at hand). This excluded diagnostic edits and edits made to undo some previous incorrect change. For each edit, we calculated the average guidance level of the TA's utterances made between the previous edit and this one. This provides an estimate of the extent to which the TA explained their thought process for the edit: talking through their thought process would mean that the TA had to provide explanations at a higher level of abstraction than simply stating what they want to change.

Figure 3a shows the histograms of the mean guidance level for edits that turned out to be correct and ones that turned out to be wrong. We used the Bayesian estimation model described in [14] to estimate the credible difference between the means of the distributions of correct and wrong suggestions made by the TA. As seen in Figure 3b, the 95% HDI of the difference is entirely positive. Therefore, we can credibly conclude that TAs' correct edit suggestions come from a distribution with a lower (less explicit) mean guidance level than suggestions that are incorrect. In other words, making a well-motivated suggestion tends to be a better idea than making one without providing much motivation.

4.2.2 Finding and diagnosing bugs. TAs often start the process of diagnosing a bug by looking at the current output of the program, e.g. an error message, an exception, a unit test failure, or a generated file. However, TA behavior differs in how and to what extent they extract information from the output. As we have seen in Section 4.1.2, sometimes the TA uses the output solely as a clue to where in the code they should start looking for the bug, for example by

going to one of the lines in a stack trace. Their main strategy in locating the bug is then to scan the code in the vicinity of that line.

This type of strategy may seem like a tempting shortcut to TAs. If they can look at the code and visually identify the problem, they don't need to spend time interpreting or considering the full context of the output. This strategy may work well enough at the beginning of the semester, when the homework problems are easier and the students' issues tend to be simpler too. However, in more complex situations, it can easily lead the debugging down the wrong path, since the TA is making much less informed decisions about what could or could not cause the current issue.

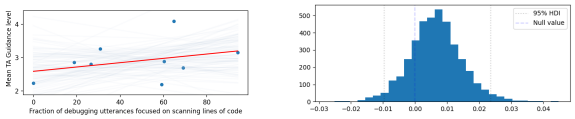
For example, one extended debugging sequence started with the TA noting that the stack trace for an exception pointed to a function which was given as part of the starter code: "on line 15? Did we not give you that code?". The exception was triggered because the student had misspelled a file name they were trying to open. Although the student and TA briefly acknowledged the error message ("no such directory"), they instead focused on reading through one of the student's functions and fixated on a totally unrelated issue of whether a variable update should be inside or outside of a loop: "And then let's just move num=1 to outside of the for loop." Once they saw that this change had no effect (since that code wasn't even being reached), they looked at the error message again and noticed the incorrect spelling.

By contrast, some TAs devote a considerable amount of focus and attention to **explicitly interpreting** the contents of the output. For example: "And let's try to think through why we're ending up with this exception ... It seems like every time we run this ... we're going straight to the except. Which means something in this try block is hitting an error like every time." Here, the TA is not able to immediately identify what is going wrong. But by reasoning about what the output could mean, they are able to narrow down not only where the problem is (in the try block), but also what they should do next to diagnose the problem ("think through why" this exception is happening).

Given this goal, the TA focused on interpreting the exception and tracing it back to its cause:

> And what it's saying is "List indices must be integers or slices, not strings" ... you think it could be a dictionary, so you're looking up with a key but it's saying, "Nope, this is a list". So our json_dict on line 158 that we're calling, that's the problem, that json_dict that we've read in is really just a list. So where that json_dict come from? line 142, when we read the JSON of the cache file name. So let's go look at read_json and see if it's returning an empty list at any point. It is... on line 43, you got an empty [list] called empty_dict. Switch those for curly braces, you'll end up with a dictionary.

By synthesizing the error message with the code's apparent intent (to write to a dictionary), the TA was able to build the understanding that a supposed dictionary object was actually a list, and then trace back the issue to where that list was created in a completely different part of the code. This issue would have been really hard to discover by simply scanning the code around the exception, without interpreting the error message.



(a) Data distribution and sample credible regression lines **(b) Histogram of the posterior distribution of the slope**

Figure 4: Correlation between fraction of debugging utterances that were focused on lines of code and the mean TA guidance level.

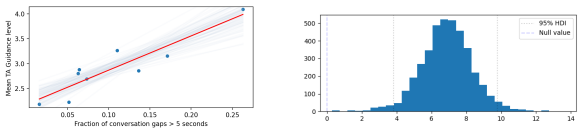
To measure the extent to which TAs focus on debugging by scanning through code, we calculated the fraction of TAs' debugging utterances that were specifically about some line or line(s) of code. We chose to measure TA focus on lines of code, rather than their focus on the output, because in two of the help sessions, nobody looked at the output at all. In those help sessions, the TA was so confident that they saw the problem that they didn't check their answers by running code. Thus, any potential debugging difficulties were obscured by the fact that they didn't try testing their changes.

Figure 4b shows that while there is a possible relationship, we cannot credibly conclude that there is a linear correlation, since the null value falls within the 95% HDI of the credible values. Part of the issue may be that the difficulty of the underlying problem is not factored into this prediction. As we discussed above, in some cases, it is actually easy for the TA to glance at a line of code and visually identify the issue. In those cases, the TA would be able to focus all of their attention on finding ways to indirectly guide the student to the answer they see.

4.2.3 Collaborating with the student. In some help sessions, the TAs actively engaged the student in the discussion, and tried to ensure that they were both on the same page. In these sessions, the TAs often actively verbalized their thought process: "We're getting some sort of JSON back because the json.loads isn't popping an error. But the JSON that we get back doesn't have a status key in it, which tells me that maybe something's going wrong there."

In other help sessions, the TA spent much less time on keeping the student in the loop, and instead preferred to study and debug the code unilaterally. These types of sessions were often characterized by the TA verbalizing lines of code as they considered them: "Okay, <li class...> so there's multiple within each... it looks like find all... " Followed by a relatively long pause and then either a next-step suggestion for the student: "Can we look up the class name starting with an underscore and then 1?" or some aside on their confusion: "Wow, this is quite difficult to know what's going on."

In these cases, the TA was largely on their own in terms of debugging, since the student had no way of knowing what they were thinking about. So, even if the student had some relevant insight, they did not have the opportunity to jump in and help debug more effectively. This solo debugging strategy was also often directly detrimental to the TA's quality of guiding and communicating with the student — when the TA discovered something helpful after long periods of thinking about code, they tended to blurt the answer out to the student instead of guiding them toward the solution,



(a) Data distribution and sample credible regression lines

(b) Histogram of the posterior distribution of the slope

Figure 5: Correlation between the fraction of long time gaps between utterances and the mean TA guidance level in that help session.

e.g. "Okay, I understand what the error message is now. You just reversed the arguments."

We used Bayesian linear regression to understand the relationship between how often there was a long (at least 5 seconds) gap in the conversation, indicating that one or both participants were likely thinking about something instead of communicating their thought process; and the overall guidance level for that help session. Figure 5a shows the result of the regression. We can see that there is a strong positive correlation between having many long gaps in the conversation and providing very explicit and direct guidance. The histogram in Figure 5b shows that the 95% HDI of the slope of this regression is quite far from 0, indicating that the relationship is significant.

4.3 Discussion

The quantitative analyses of these TA strategies use inexact numeric proxies to measure each strategy. This is in large part because we had defined our coding scheme before analyzing the data to identify these strategies. So, the coding scheme sometimes lacks nuance that could have been helpful in quantifying these TA strategies. For example, if our coding scheme included identifying whether an utterance provided an **interpretation** of an output message, we could more precisely capture to what extent the TA was using an output-focused strategy for identifying bugs.

Nevertheless, these proxies suggest that a TA's choice of strategy is related to the success they have in both debugging and guiding the student. Moreover, these results continue to reinforce the conclusion that the difficulties a TA has debugging and guiding are closely and positively related to each other. Yet prior research indicates that TAs often perceive a **tension** between helping the student solve their issue and guiding the student in a way that helps them learn [24, 28, 30]. In other words, they perceive a **trade-off** between focusing on debugging and focusing on guiding. Thus, it may be worthwhile to emphasize to TAs that certain strategies can make both aspects easier in tandem.

These relationships are not necessarily causal. For example, it could be that a TA is more likely to make an erroneous suggestion if they haven't reasoned it through at a high level; or it could be that a TA is less likely to talk through their reasoning if they don't *have* a clear line of reasoning about their suggestion, which also makes the suggestion less likely to be correct. However, the set of strategies we've identified can lead to a set of actionable guidelines regardless of causality. "If you can't explain it, avoid suggesting it" could be an actionable guideline in both of the possible cases in the example above. Moreover, guidelines that focus on concrete debugging strategies may be more actionable and constructive than simply asking TAs to follow good tutoring practices. Our results suggest that TAs do already try to follow good tutoring practices. For example, if TAs weren't already aware of the benefits of asking leading questions, they wouldn't ask significantly more of them when they don't have to spend as much time debugging.

5 Limitations

The main limitations of this study are the small sample size of participants and somewhat limited scope. This study involved four TAs and seven students from one specific beginner-to-intermediate programming class at one university, interacting across nine different office hour sessions. Although we were able to draw statistically significant conclusions, it is not clear how these conclusions about TA behavior might generalize to different classes in different contexts. For example, in a similar previous study conducted in a more advanced class at a different university [23], the TAs did very little debugging in general. Therefore, it is unlikely that their debugging strategies had the same effect on their ability to guide the students. More research is needed to understand how TA behavior varies across different contexts.

Moreover, the recordings we analyzed were, to some extent, self-selected by the TAs we were analyzing. Although in theory all TAs who agreed to participate in the study were asked to record and upload all of their help sessions with students who were also participating, this clearly did not happen in practice. In particular, only four out of the seven TAs who agreed to participate uploaded any recordings at all. Part of the reason for this could be that recording, tracking, and uploading videos of help sessions was too much overhead for TAs during office hours. Another reason could be that TAs felt self-conscious recording and uploading their help sessions, especially when the TA felt they didn't perform well. Thus, the recordings we analyzed may not be a representative sample of the types of help sessions that occurred. However, these recordings still presented a wide range of TA behaviors and types of difficulties encountered.

6 Conclusion

We demonstrated a clear positive correlation between the amount of difficulty a TA has **debugging** student code, and the difficulty they have in using good tutoring practices to **guide** the student toward understanding the issue.

Through a qualitative analysis of two contrasting sessions conducted by the same TA, we observed that a TA's choice of debugging strategy can affect how well they guide the student. Certain debugging strategies may seem like shortcuts when the student's problems are not too difficult for the TA. But in more complex situations, these same strategies can lead to pitfalls that affect both their ability to debug the code and their ability to effectively guide the student.

We identified three specific dimensions of debugging strategies where the TA's choices can have an impact on both debugging and guiding. We found both qualitative and quantitative evidence that these choices in debugging strategy affect how well the TA guides the student within a particular help session.

Finally, we discussed the implications of these findings on the types of guidelines that may help TAs both debug code and guide students more effectively. Specifically, we discussed that guidelines which focus on concrete debugging strategies, and explicitly emphasize how these strategies could make both debugging and guiding easier at the same time, may be a constructive way to lead TAs toward better practices in both areas.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 2214538

References

[1] 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*. 9–13.

[2] Rebecca Brent, Jason Maners, Dianne Raubenheimer, and Amy Craig. 2007. 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*. IEEE, F1J–19.

[3] Alan Y. Cheng, Ellie Tanimura, Joseph Tey, Andrew C. Wu, and Emma Brunskill. 2024. Brief, Just-in-Time Teaching Tips to Support Computer Science Tutors. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 200–206. https://doi.org/10.1145/3626252.3630794

[4] Jennifer G. Cromley. 2005. What Do Reading Tutors Do? A Naturalistic Study of More and Less Experienced Tutors in Reading. *Discourse Processes* 40, 2 (Sept. 2005), 83–113. https://doi.org/10.1207/s15326950dp4002_1

[5] 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 SIGCSE technical symposium on Computer science education*. 46–50.

[6] 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 *2017 IEEE Frontiers in Education Conference (FIE)*. 1–6. https://doi.org/10.1109/FIE.2017.8190650

[7] Sue Fitzgerald, Gary Lewandowski, Renee McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education* 18, 2 (2008), 93–116. Publisher: Taylor & Francis.

[8] 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*. 658–663.

[9] Olivier Goletti, Kim Mens, and Felienne Hermans. 2022. An Analysis of Tutors' Adoption of Explicit Instructional Strategies in an Introductory Programming Course. In *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research (Koli Calling '22)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3564721.3565951

[10] Irvin R. Katz and John R. Anderson. 1987. Debugging: An analysis of bug-location strategies. *Human-Computer Interaction* 3, 4 (1987), 351–399. Publisher: Taylor & Francis.

[11] Matthew Kay, Gregory L. Nelson, and Eric B. Hekler. 2016. Researcher-Centered Design of Statistics: Why Bayesian Statistics Better Fit the Culture and Incentives of HCI. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 4521–4532. https://doi.org/10.1145/2858036.2858465

[12] Sophia Krause-Levy, Rachel S. Lim, Ismael Villegas Molina, Yingjun Cao, and Leo Porter. 2022. An Exploration of Student-Tutor Interactions in Computing. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1 (ITiCSE '22)*. Association for Computing Machinery, New York, NY, USA, 435–441. https://doi.org/10.1145/3502718.3524786

[13] John Kruschke. 2014. Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan. (2014). https://books.google.com/books?hl=en&lr=&id=FzvLAwAAQBAJ&oi=fnd&pg=PP1&dq=kruschke&ots=ChqhNXyfXM&sig=vtn5IUxqV5j15NJvJpqu7pa_Iso Publisher: Academic Press.

[14] John K. Kruschke. 2013. Bayesian estimation supersedes the t test. *Journal of Experimental Psychology: General* 142, 2 (2013), 573. https://psycnet.apa.org/journals/xge/142/2/573/ Publisher: American Psychological Association.

[15] John K. Kruschke, Herman Aguinis, and Harry Joo. 2012. The Time Has Come: Bayesian Methods for Data Analysis in the Organizational Sciences. *Organizational Research Methods* 15, 4 (Oct. 2012), 722–752. https://doi.org/10.1177/1094428112457829

[16] Kyungbin Kwon, Christiana D. Kumalasari, and Jane L. Howland. 2011. Self-Explanation Prompts on Problem-Solving Performance in an Interactive Learning Environment. *Journal of Interactive Online Learning* 10, 2 (2011). https://www.academia.edu/download/34736057/Kwon_2011_ex.pdf

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

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

[19] Rachel S. Lim, Sophia Krause-Levy, Ismael Villegas Molina, and Leo Porter. 2023. Student Expectations of Tutors in Computing Courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 437–443. https://doi.org/10.1145/3545945.3569766

[20] Zhongxiu Liu, Rui Zhi, Andrew Hicks, and Tiffany Barnes. 2017. Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education* 27, 1 (2017), 1–29.

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

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

[23] Yana Malysheva, John Allen, and Caitlin Kelleher. 2022. How Do Teaching Assistants Teach? Characterizing the Interactions Between Students and TAs in a Computer Science Course. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–9. https://doi.org/10.1109/VL/HCC53370.2022.9832962 ISSN: 1943-6106.

[24] 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*. ACM, Virtual Event USA, 502–508. https://doi.org/10.1145/3408877.3432533

[25] 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*. ACM, Baltimore Maryland USA, 122–127. https://doi.org/10.1145/3159450.3159601

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

[27] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: the good, the bad, and the quirky – a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin* 40, 1 (March 2008), 163–167. https://doi.org/10.1145/1352322.1352191

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

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

[30] Emma Riese, Madeleine Lorås, Martin Ukrop, and Tomáš Effenberger. 2021. Challenges Faced by Teaching Assistants in Computer Science Education Across Europe. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. Association for Computing Machinery, New York, NY, USA, 547–553. https://doi.org/10.1145/3430665.3456304

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

[32] Lasang Jimba Tamang, Zeyad Alshaikh, Nisrine Ait Khayi, Priti Oli, and Vasile Rus. 2021. A Comparative Study of Free Self-Explanations and Socratic Tutoring Explanations for Source Code Comprehension. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. ACM, Virtual Event USA, 219–225. https://doi.org/10.1145/3408877.3432423

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

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

[35] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Jaakko Kurhila. 2013. 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*. 123–128.

[36] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2023. A Think-Aloud Study of Novice Debugging. *ACM Transactions on Computing Education* 23, 2 (June 2023), 28:1–28:38. https://doi.org/10.1145/3589004