Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs

Julia M. Markel
UC San Diego
jmarkel@ucsd.edu

Philip J. Guo
UC San Diego
pg@ucsd.edu

ABSTRACT

As CS enrollments continue to grow, introductory courses are employing more undergraduate TAs. One of their main roles is performing one-on-one tutoring in the computer lab to help students understand and debug their programming assignments. What goes on in the mind of an undergraduate TA when they are helping students with programming? In this experience report, we present firsthand accounts from an undergraduate TA documenting her 36 hours of in-lab tutoring for a CS2 course, where she engaged in 69 one-on-one help sessions. This report provides a unique perspective from an undergraduate's point-of-view rather than a faculty member's. We summarize her experiences by constructing a four-part model of tutoring interactions: a) The tutor begins the session with an initial state of mind (e.g., their energy/focus level, perceived time pressure). b) They observe the student's outward state upon arrival (e.g., how much they seem to care about learning). c) Using that observation, the tutor infers what might be going on inside the student's mind. d) The combination of what goes on inside the tutor's and student's minds affects tutoring interactions, which progress from diagnosis to planning to an explain-code-react loop to post-resolution activities. We conclude by discussing ways that this model can be used to design scaffolding for training novice TAs and software tools to help TAs scale their efforts to larger classes.

CCS CONCEPTS

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

KEYWORDS

Lab Tutoring, Peer Tutoring, Undergraduate Teaching Assistants

ACM Reference Format:
Julia M. Markel and Philip J. Guo. 2021. Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432533

1 INTRODUCTION

As CS enrollments have grown over the past few decades, universities have been hiring more undergraduates as teaching assistants. For instance, large CS courses often employ dozens of undergraduate TAs (called UTAs) to serve up to a thousand or more enrolled students [2]. Mirza et al. surveyed 40 papers about CS undergraduate TAs and found that their most common task was *"assisting students in labs on programming assignments"* [22]. Each UTA usually holds weekly tutoring hours in the computer lab; students come there to work on assignments and put themselves on a queue to get help. These sorts of peer tutoring [11] interactions are critical for providing both technical and emotional support. As fellow undergrads who have recently taken the same course, UTAs can be more relatable to students than graduate students or faculty are [33].

Despite the pervasiveness of lab tutoring in CS courses, to our knowledge no prior work has reported on the interpersonal dynamics of this kind of help-giving interaction from a tutor's perspective. Lab tutoring is a cognitively complex yet ill-understood form of teaching: UTAs must quickly jump in to understand each student's problem without prior context, figure out the best way to assist without giving away the solution, provide emotional reassurance, and balance quality of support versus the pressing time constraints of needing to help other students who are waiting on the queue.

What goes on in the mind of an undergraduate TA when they help students with programming problems in the computer lab? To address this question, we present an experience report where the first author, a 20-year-old female computing student, documented 36 hours of her tutoring experiences as a UTA for a CS2 course where she engaged in 69 one-on-one help sessions.

We report her firsthand experience and then distill it into a four-part model shown in Figure 1: a) The tutor begins the session with an initial state of mind (e.g., energy/focus level). b) They observe the student's outward state upon arrival (e.g., initial demeanor). c) Using that observation, they *infer* what might be going on inside the student's mind. d) The tutor's and student's combined states of mind affect the dynamics of tutoring interactions, which consist of five phases: arrival, diagnosis, game plan, explain-code-react loop, and post-resolution activities. This model can potentially help researchers to design policies and tools to support TAs in providing one-on-one tutoring for students in a way that scales to meet ever-growing enrollments. The contributions of this paper are:

- An undergraduate TA's (UTA's) firsthand experience of 36 hours of lab tutoring for an introductory CS course.
- A set of challenges that UTAs face when tutoring in the lab.
- Design ideas for policies and tools to support UTA tutoring.

2 RELATED WORK

To our knowledge, we are the first to model the experiences of a CS undergraduate TA while they are tutoring in the computer lab. Prior work has interviewed UTAs [25, 31], observed them working

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432533

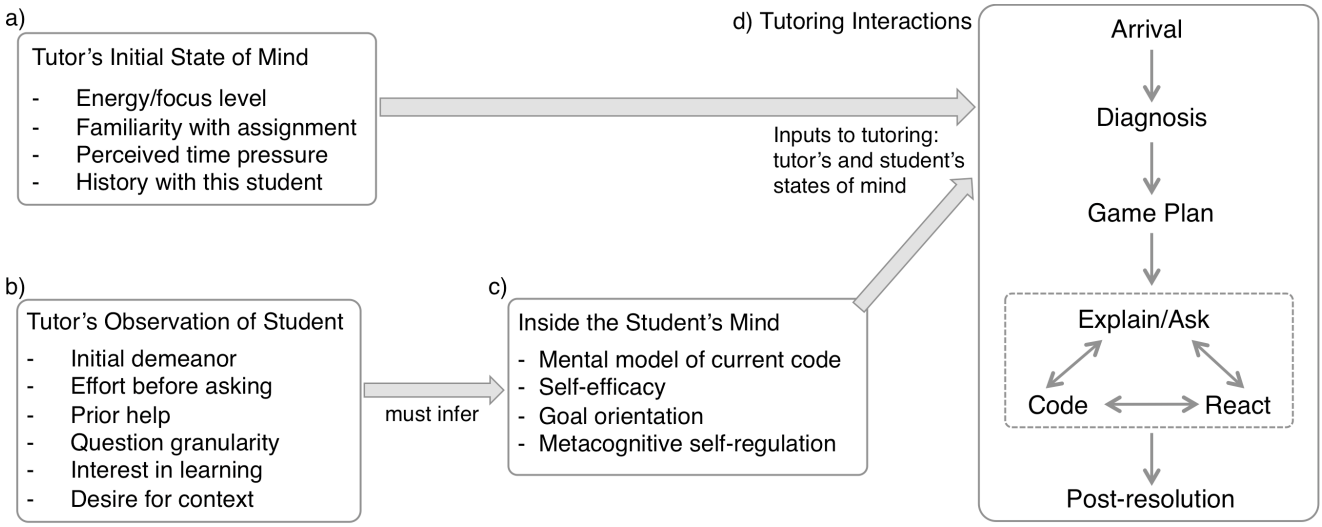

Figure 1: A model that summarizes the first author's experiences helping students in the lab as an undergraduate CS TA.

in the lab [25], and presented a self-reflection of overall UTA experiences [12]. Those works focus more on higher-level perceptions of job satisfaction and challenges rather than the detailed step-by-step interpersonal dynamics of one-on-one tutoring that we focus on.

Mirza et al. surveyed 40 research papers about undergraduate TAs (UTAs) [22] and found that tutoring in lab hours was the most common UTA duty, with the next two most common being leading sections and grading. Despite this fact, none of the papers they surveyed investigated what occurs during lab tutoring interactions. The closest lines of research presented the kinds of questions that students asked in UTA office hours [30], interviews with students and UTAs about post-hoc perceptions of lab tutoring [8], student perceptions of peer tutors [13], impact on grades [10, 27], training materials and competence models for UTAs [7, 9, 32], and student perceptions of getting help from a human tutor versus an ITS (intelligent tutoring system) in a controlled study [28]. Our contribution to this literature is a firsthand account of what goes through a UTA's mind as they are in the midst of tutoring.

Our work also complements studies that investigate emotional barriers that students face in introductory programming, such as Kinnunen and Simon's series of studies on the emotional toll and self-efficacy impacts on students struggling with CS1 programming assignments [16–18], the role of metacognitive self-regulation in programming problem solving [20, 24], and issues of anxiety related to math and computing [23]. More broadly, there is a rich literature on technical misconceptions that students have about programming languages and concepts [29]. Our experience report augments these lines of work with a model of what lab tutors do to help students overcome these emotional and technical barriers.

3 SETTING: LAB TUTORING IN CS COURSE

The first author is a 3rd-year undergraduate computer engineering major (female, 20 years old) at a large public U.S. university. She excelled in a CS2 course (details below) during her first year and was recruited to be an undergraduate TA (UTA) for that course. Over the past two years, she has been a UTA for 6 quarter-long terms and is now one of the *head UTAs* who mentors junior UTAs.

The site for her field observations was an offering of a CS2 course, which enrolled ~200 students and lasted 10 weeks from January to March 2020. This course is required for all computer science and computer engineering majors and covers the implementation of data structures in C, C++, and Java. Each week, students must individually complete a programming assignment that involves implementing progressively more complex data structures (e.g., stacks, linked lists, binary trees, hash tables) based upon staff-provided starter code. These assignments are especially challenging because students need to implement each assigned data structure in *two* different languages (starting with C and Java, then C++ and Java).

To help students on these programming assignments, the UTA staff signs up for shifts to work as tutors in the computer lab. At any time there are usually 2 to 4 tutors in the lab. Students come to the lab space to work at their own pace on Linux desktop computers there or on their own laptops. To request help from a tutor, they use a web application to put themselves on the help queue.

The first author worked a single 4-hour shift in-person in the computer lab every Saturday (9am–1pm) for 9 weeks during the winter 2020 term; the 10-week term ended in early March 2020 right before COVID-19 quarantines began. She held 69 total tutoring sessions throughout the term (average of 7.66 per weekly shift), with each session lasting 22 minutes on average.

The first author served as a participant-observer for this study. She took field notes for a few minutes immediately after helping each student and before moving onto the next one in the queue. Each note aimed to capture both the technical and affective (emotional) details of the interaction, which include the context of the student's question, observations of the student's affective state before and during the interaction, her own affective state, technical details of the problem and debugging process, and what happened immediately after the issue was (or was not) resolved. After each weekly lab shift, she met with a faculty research advisor (who was *not* involved in teaching this course) to analyze her field notes and

iteratively construct a model that summarizes her tutoring interactions. To do so, we took an inductive analysis approach [5] to distill a set of higher-level themes from specific experiential details.

4 TUTORING EXPERIENCES AND MODEL

We now present all parts of the model in Figure 1. Since this is an experience report, we will use "I" throughout this section to convey the first author's experiences that we used to construct this model.

4.1 Tutor's Initial State of Mind

When I take a student's help request from the queue and approach them in the lab, my state of mind at that time affects how the tutoring interaction will proceed. Figure 1a shows four salient factors:

Energy/focus level: Ideally each tutor comes into the lab fully-focused on the task at hand, but since tutors are also students (fellow undergrads in the case of UTAs), their energy and focus levels vary with the usual rhythms of student life. For instance, if I did not sleep well or have my own homework due soon, that will make me less focused during my lab shift. Taking breaks between students can help me recharge, but that is often hard for me to justify if there are many students waiting on the queue for their turn.

Familiarity with assignment: Each tutor should come into the lab fully-familiar with the programming assignment for that week, having gone over it in detail beforehand. But again, since tutors are students with their own coursework to manage, they often do not have time to do extensive prep. I personally have a cold-start at the beginning of each lab shift and actually re-familiarize myself with the assignment *while* I am helping the first few students. But then as my shift progresses, I incrementally get more familiar with the assignment, having loaded the technical details into my brain's cache and recalling common issues brought up by prior students. Then as my shift progresses, even though I grow progressively more fatigued, I also grow more familiar with assignment details, which somewhat compensates for my lower energy/focus level.

Perceived time pressure: Ideally each tutor treats all student interactions with the same level of care, but in reality time pressure affects how much they can focus on any one student. I personally feel time pressure mainly based on the current size of the help queue; it also happens when helping a student toward the end of my shift and needing to decide whether I want to stay beyond my posted hours. When the queue is relatively empty, I feel less time pressure and can more comfortably go in-depth with each student. However, I never feel fully relaxed since I always have the lingering "calm before the storm" feeling that the queue size could spike any time as more students arrive. When the queue is moderate (e.g., 3 to 5 students), I still feel comfortable staying with a student for as long as it takes, since I know that other tutors are in the lab tending to the queue. However, when the queue is very long, then I feel intense pressure to be as fast as possible, at the expense of quality.

History with this student: I try to treat all students similarly, but there are some "regulars" who return to lab week after week during my shifts. In the best case, I am more invested in those students whom I see regularly and have developed good rapport with. Even though lab tutoring is fairly "transactional" and not meant to be a long-term mentoring relationship, I take pride in seeing those students grow throughout the term; I remember that they were visibly appreciative of my earlier help in prior weeks, so I want to continue being as helpful as possible. In the worst case, for the few students whom I did not have good rapport with in the past, I may actively avoid taking their question; one way to do so is to "wait it out" by helping another student for longer and hoping that another tutor frees up to take that student from the queue.

4.2 Tutor's Observation of Student

Aside from my own state of mind, what I observe when I first approach a student in the lab also impacts how tutoring will proceed. Figure 1b shows the six characteristics that I notice most:

Initial demeanor: I first notice the student's outward demeanor, which varies along two dimensions: quiet to outspoken, and insecure to confident. While those could be correlated (e.g., outspoken students being more confident), I have seen all four combinations in the lab. Many are indeed quiet because they seem insecure about their programming skills. But some are quiet even though they are highly confident in their progress and only need light help. Conversely, some outspoken students appear to be hiding insecurities about their knowledge gaps, which I soon discover.

Effort before asking: Another first impression I get is how much effort the student seems to have put into the assignment before asking for help. I see indicators of effort by the materials they show me when I approach. The most salient is the current state of their code; some simply have the instructor-provided starter code on display without much code of their own written yet, while others have substantial amounts of their own code interleaved with print-debugging statements. Some also have lots of web browser tabs open, indicating that they have been seeking help online, or even sketches of tracing through code on paper; both indicate high effort.

Prior help: Related to effort, some students tell me that they have already gotten help on their problem from friends or from another tutor. Although prior help can be beneficial, it may also confuse the student more if that help was not well-scoped for their needs.

Question granularity: Some students can precisely phrase their questions to pinpoint exactly where they are struggling (e.g., down to a particular function or line of code), even though their initial guesses of the problem location may be incorrect. In contrast, other students ask vague questions like "how do I start on this part?" Question granularity often depends on which stage of the programming assignment [21] they are currently attempting: e.g., planning their approach to the problem, trying to get their initial code to compile, diagnosing run-time errors, or making their code more robust to tricky edge cases once it already works on an initial set of tests. Granularity can vary even within a stage. For instance, when asking about a run-time error, they can either ask "why is my code crashing?" (vague) or "I think this part of my code is throwing an out-of-bounds exception but I'm not sure why" (specific).

Interest in learning: I can usually sense a student's interest in learning by how they phrase their questions. On one end, some ask "what's wrong with this?" or "how do I fix this?" with a curt tone, which indicates that they simply want a quick answer without caring as much about deeper understanding. Bolder students may even probe for answers with questions worded like "so, how would

I write this part?" rather than first making an earnest effort and then asking me to check their work. On the other end are students who demonstrate a genuine interest in learning the underlying concepts and not just passing the assignments. For instance, some ask questions like "I tried X and thought it would work this way, but why doesn't it?" Other students have code that works fine but they want to have me walk through it with them so they can better understand *why* it works the way that it does.

Desire for context: Related to interest, another trait that varies between students is their desire for understanding the context surrounding their question. Since programming assignments often contain dozens of lines of code, much of it in instructor-provided starter code and API calls, some students want to understand how the surrounding code operates and connects with the code that they are writing. While that might appear to be a good sign (it shows interest in learning), too much of a desire for context can be counterproductive: for instance, it is hard for me to justify spending extra time explaining context that is not as relevant to the assignment when others are waiting to get help, especially when the queue is long. On the other end, some students are too myopically focused on just the lines of code they are debugging at the moment and have no desire to "zoom out" to learn more context.

4.3 Inferring What is Inside the Student's Mind

Based on my observations I must then infer what might be going on inside the student's mind (Figure 1c). One central challenge that tutors like me face is that *they must build a mental model of what is going on inside the student's mind* without being able to directly observe that internal state. Most importantly, I must infer the student's (faulty) mental model of the current code they are debugging, based on how they phrase their question. I must also infer their emotional state, which includes three components: 1) *Self-efficacy* [3] represents how much the student believes that they are capable of solving the given programming assignment. 2) *Goal orientation* [26] represents whether the student's goals are more geared toward simply completing the assignment (i.e., performance-oriented) or toward more deeply learning the underlying course material (i.e., mastery-oriented). 3) *Metacognitive self-regulation* represents the student's awareness of their own progress and limitations, along with their ability to seek out help at the proper times [4, 6, 21].

From Figure 1b, my observation of the student's *initial demeanor* is the most direct indicator of their self-efficacy. Another indicator is *question granularity*: students with higher self-efficacy are usually able to more precisely pinpoint the kind of question they want to ask the tutor. The *interest in learning* and *desire for context* that they exhibit both potentially indicate their goal orientation (performance vs. mastery oriented). Finally, their amount of *effort before asking* for help and whether they have already gotten *prior help* are both signs of metacognitive self-regulation in terms of knowing how to plan their progress and to ask for help at the necessary times.

4.4 Tutoring Interactions

As Figure 1d shows, my tutoring interactions contain five phases: arrival, diagnosis, game plan, explain-code-react, and post-resolution.

4.4.1 Arrival. When I arrive to greet a student, I adopt a demeanor that ranges from relatable to authoritative, depending on both my own mental state and my observation of the student's state (see prior sections). For most students, I lean on the side of being relatable and accommodating since I want to give them emotional reassurance in addition to technical support. Also, I want to establish upfront that they should not be embarrassed to ask their question. However, when my energy level is low or I am under time pressure, I may adopt a firmer demeanor where I focus more on the technical issue at hand and not as much on emotional reassurance. At the most extreme, when I have not had a good history with this particular student or can sense that they may question my expertise, then I am the most firm in establishing myself as authoritative.

4.4.2 Diagnosis. Having calibrated my demeanor, my first task is to work together with the student to diagnose their underlying problem. The easiest case is when I have seen that same problem before, so I can eyeball the issue by just looking at their code and terminal output. The next easiest case is if the student articulates their question clearly and their code is clean; even if I have not seen this problem before, I am confident that we can diagnose it together successfully. What I usually do here is ask the student to trace through the relevant code and verbally tell me what they think is going on at each step (e.g., I ask "what do you think this line does?" or "what happens next?"). Through this questioning process, I can usually tell where and what their misconception is.

More often, the student's question is less well-defined or even inadvertently misleading, or they may not be able to reproduce the bug reliably. In those cases, I trace through the code myself, draw out values on paper, or tell them to insert print statements to verify my hypotheses. If there is a segfault or other crash, I may also use a debugger to quickly find the crashing location and stack trace.

The hardest cases to diagnose are when the student's code is extremely messy and confusing. Those students also have the hardest time clearly explaining their problem to me. I need to expend a lot of energy on just understanding their code, which leaves me with less energy to help them fix it. I also sometimes get flustered as I attempt to diagnose difficult bugs. When that happens, I need to try to keep a stoic "poker face" so that I do not inadvertently make the student feel bad that their code is hard to understand.

4.4.3 Game Plan. After a successful diagnosis[1], I formulate a plan for the rest of the session. This requires careful thought because the purpose of tutoring is *not* to simply fix the student's problem; it is to teach the student something meaningful so that the next time they encounter a similar problem, they can fix it themselves. Depending on my rapport with the student, I will formulate one of three types of game plans: 1) *Fix-then-explain*: When the bug is simple, I will directly suggest a hint that leads to a fix. Once the student has resolved the problem, then I explain why it works. The advantage here is that the student feels relieved that their problem is resolved quickly and then may be more willing to listen to an explanation. 2) *Interactive*: Most of the time, we will engage in an interactive conversation where I incrementally nudge them closer to the bug fix by asking them to explain their mental model, clarifying any misconceptions, then giving them some relevant code hints.

[1] If I cannot diagnose a bug, I will usually ask another TA or post on the class forum.

3) *Explain-then-fix*: In rare cases, I will explain upfront what their misunderstanding is and exactly what we are going to do to fix it. Here I feel like a doctor explaining a diagnosis and treatment to a patient. I take this approach when I take an authoritative demeanor, such as with a student who may question my competence.

4.4.4 Explain-Code-React Loop. After formulating a game plan, the student and I enter into the main loop that oscillates between me explaining and asking questions (*explain/ask*), working with them to modify their code (*code*), and assessing their reaction (*react*). Depending on the game plan (see above), all of these actions may be interleaved in any order throughout the tutoring session (e.g., explain then react then code, or code then explain then react).

Explain/Ask: Depending on the student's interest in learning and desire for context, along with time constraints, I will either adopt a more *direct instruction* approach of explaining concepts to them or a more *constructivist* approach of asking questions to get them to discover the knowledge for themselves [19] (usually a mix of both).

Code: Since I do not want to give away the solution code or even touch the student's keyboard, I must carefully work with them to guide them toward a viable solution while keeping their morale up throughout the (sometimes-long) process. The main consideration here is *code preservation* – how much of the student's existing code to preserve versus getting them to start over from scratch.

I always try to preserve the student's original code, especially if I can tell that they have already put in a significant number of hours and formed an emotional connection with it. Even though there are always more optimal ways for an expert to write that code, it is demoralizing for a student to have a tutor invalidate their approach. So I try my best to work with what is in front of me.

In the most extreme case, when I feel like it is best for a student to start over from scratch, I still never tell them "you should start over" since that feels too harsh. Instead I take a Ship of Theseus [1] approach and guide them to incrementally rewrite each portion until the result is that their code is nearly brand-new.

React: As I am explaining, asking questions, or guiding the student's coding, I constantly gauge their reactions. Student reactions vary along two dimensions: engagement level and demeanor.

Students react differently based on their engagement level: On one end, some students get distracted by their phone's notifications or start texting on it while I am helping them. Since I sense that they are not putting in effort at the moment, I try to point them toward the right direction and then move on to the next student. In the middle, most students passively listen to my explanations and will try what I suggest, but they will not write code unless I prompt them. The most common passive reaction is a silent nod, which is hard for me to interpret. I need to ask further questions, watch their facial expressions, give them time to react, and pay close attention to how they are speaking to determine whether a nod means they truly understand or whether they are just being polite. Next, more active students will take the initiative to pull out additional notes or make drawings themselves on paper instead of just watching me draw. The most engaged students will take the initiative to write code and think-aloud about their process.

Students also react to me differently based on their demeanor throughout the session. Most remain calm but some gradually grow more agitated when they realize that their misunderstanding is deeper than they originally expected. Even though those students are more agitated, I can tell that they genuinely want to work with me to understand and solve their problem. But this change in demeanor can cause me to get flustered and lose confidence in my ability to help. When that happens, I try to backtrack to the prior "checkpoint" where they felt comfortable, reestablish a sense of calm, and then gradually work forward toward the frustrating part. However, some get agitated because they just want me to give them the solution; they might cut me off in the midst of explaining something or ask me leading questions to fish for answers.

4.4.5 Post-resolution. After resolving the student's issue, I rarely just leave right away unless: a) the resolution was straightforward (e.g., a simple syntax error) so it requires no further explanation, b) the student shows little interest in learning or seems too exhausted, or c) the queue is long so I feel compelled to move on.

I typically stay with the student for a few minutes post-resolution because I feel like that is the best *teachable moment* [14] when they are the most receptive to learning. We have just gone on a journey together to a successful end, and they often have follow-up questions or want to reflect on it with me. The ideal I strive for when tutoring is leaving students with more generalizable knowledge about the subject so that they can figure out how to solve similar problems on their own in the future. If I am just providing temporary fixes and not teaching them how to debug on their own, then they will be just as stuck the next time a similar problem occurs.

I also try to provide emotional reassurance at this time by acknowledging that this bug was indeed difficult and that they did a good job working through it with me. At this point some students ask me how long I spent on this assignment back when I took the class, so hearing me reflecting on my own effort and struggles gives them further reassurance that the amount of time they spent is normal. This is a relatability benefit to being an undergraduate TA, since I was in the student's shoes just a year or two earlier. Right before leaving, I give them encouragement that the path is now clear for them to make headway on the next part of the assignment.

5 DISCUSSION: LESSONS LEARNED

Here we present some lessons learned from this process of reflecting on UTA tutoring experiences. Our reflections revealed challenges that tutors face and potential design ideas for overcoming them.

5.1 Challenges That Programming Tutors Face

Reading the student's mind: The central challenge that tutors face is building an accurate mental model of what the student's mental model of their code is (Section 4.3). Students are inexperienced at phrasing their questions in a clear way, and their guesses at what might be wrong with their code are often incorrect.

Emotional regulation: Tutoring is a cognitively and emotionally demanding task. Under time pressure, the tutor needs to diagnose the problem and guide the student toward a fix while reacting to the student's changing demeanor. The tutor also needs to keep a calm demeanor themselves (e.g., a "poker face") so as not to inadvertently discourage the student further. Finally, since tutors are often also

students, they must regulate their emotions even when they are tired or anxious due to the other responsibilities of student life.

Maintaining student engagement: This is a universal challenge of teaching in general, and it can even manifest in a one-on-one tutoring setting. At worst, students get distracted in the midst of getting helped, but even if they try hard to pay attention, they are often passively listening and nodding. How can we get students to more actively engage while respecting the tutor's time constraints?

Teaching vs. bug-fixing: From a pedagogical standpoint, the ideal to strive for in tutoring is actually teaching the student some generalizable knowledge that they can apply to future problems. However, most students just want help getting their current code bug fixed so they can proceed with the assignment. How can tutors juggle those two competing goals while under time constraints?

Triage and prioritization: Lab tutoring works on a first-come first-served basis, enforced by the help queue. Although we should not discourage students from asking questions, the reality is that not all students are equally prepared to ask well-formed questions. Due to having to obey first-come first-serve, tutors sometimes get occupied for a long time by an unprepared student when the next student on the queue is more "ready" to get helped because they have put in more prior effort. How can we nudge students toward the "optimal" time to ask for help – not too soon and not too late? Ideally tutors can prioritize students by how ready they are at the moment while giving quick nudges to the other students to try things out on their own before accessing the tutor's scarce time.

Preserving student code: Fixing one's own buggy code can be challenging enough, but it is even harder to guide a student toward fixing *their* code while preserving as much of that code as possible. Since the tutor should not take over the student's keyboard to directly write code on the student's computer, they must essentially "write" the code in their heads first before suggesting possible next steps. Note that the tutor cannot use their own laptop since they do not have direct access to the current state of the student's code.

Getting real-time feedback: Even though UTAs can be more relatable to students than faculty are, there is still a power imbalance during tutoring sessions. This can make it hard for tutors to get real-time feedback on whether their explanation is confusing; many students will just politely nod, even if they do not fully understand. Very rarely will a student directly tell a tutor that their explanation is inadequate, so it is up to the tutor to read the student's expressions, all while trying to diagnose and explain the underlying problem.

5.2 Design Ideas for Improving Lab Tutoring

Easing problem diagnosis: While students are waiting on the queue, they could interact with an automated system such as a chat bot to help them narrow down their question. That way, when the tutor arrives, the student will more likely have formulated a more precise and targeted question, which can lead to an easier diagnosis. In the best case, reminiscent of ELIZA [34] or rubber duck debugging [15], the student actually figures out and resolves the problem on their own through the act of verbalizing their assumptions.

Encouraging deeper learning: Ideally students come out of tutoring with a deeper knowledge of underlying concepts so they can solve similar problems on their own next time. But all too often, they just want their code fixed for the practical purpose of passing that specific part of their assignment. One lightweight way to nudge students toward deeper learning is to have them write a brief structured self-reflection after they get tutored to explain their underlying misunderstanding and how that was resolved. These self-reflections can both help the student solidify their own learning and also help their classmates see what common misunderstandings arose in the assignment. Right now since both the tutor and student are in a rush to move on after tutoring is done, no lasting artifacts come out of these information-rich interactions.

Mitigating power imbalances: There will always be power imbalances between students and course staff. To mitigate possible negative effects, first it is important to train tutors to be aware of power dynamics, implicit biases, and sociocultural factors that affect students' emotional states when asking for help. Next, providing a systematic checklist for how tutors should approach each interaction step-by-step (e.g., based on Figure 1) may make the experience more standardized and thus more predictable to students. Finally, online tutoring via text chat in a web-based IDE could further reduce power imbalances because students may feel less intimidated online than if they were physically sitting next to a tutor in the lab; they could also choose to get help anonymously.

Training novice tutors: Although we did not directly study novice tutors, the first author is a head UTA who has tutored for six terms and now informally mentors new tutors. As UTAs graduate each year, it is important to constantly be training the next generation to serve as effective tutors, especially as class sizes grow. One way to train novice tutors is to pair them up with a more experienced tutor; that way, the novice can either watch the experienced tutor work, or have the experienced tutor watch and critique them. A more scalable approach is to use our model in Figure 1 to develop a structured checklist for tutors to follow when approaching each student. Our model essentially encapsulates an experienced tutor's workflow. A complementary approach is to have new tutors fill out a self-reflection form after each student interaction to summarize what they felt the main challenges and lessons were, in order to encourage them to engage in metacognition about their tutoring.

6 CONCLUSION

In this experience report we summarized the workflow and challenges that an undergraduate TA faces when tutoring in the computer lab for a 10-week CS2 course. We distilled her experiences into a four-part model that captures the tutor's and student's states of mind, along with the interpersonal dynamics of their tutoring interactions. The main limitation of this work is that since this model is based on only one UTA's experiences, more follow-up research is required to make sure that it generalizes beyond our own setting. As CS enrollments continue to grow in both in-person and online courses, we believe that recruiting, sustaining, and supporting UTAs and other peer tutors will be critical for making sure every student gets the help that they need. We hope that our initial model can start a conversation about how to best provide support to growing cohorts of UTAs via both policies and software tools.

Acknowledgments: This material is based upon work supported by the National Science Foundation under Grant No. NSF IIS-1845900.

REFERENCES

[1] [n.d.]. The Ship of Theseus: The Philosophy Foundation. https://www.philosophy-foundation.org/enquiries/view/the-ship-of-theseus. Accessed: 2020-08-01.

[2] [n.d.]. UC Berkeley: CS 61A: Structure and Interpretation of Computer Programs. https://cs61a.org/. Accessed: 2020-08-01.

[3] Albert Bandura. 1977. Self-efficacy: toward a unifying theory of behavioral change. *Psychological review* 84, 2 (1977), 191.

[4] Susan Bergin, Ronan Reilly, and Desmond Traynor. 2005. Examining the Role of Self-Regulated Learning on Introductory Programming Performance. In *Proceedings of the First International Workshop on Computing Education Research* (Seattle, WA, USA) *(ICER '05)*. Association for Computing Machinery, New York, NY, USA, 81–86. https://doi.org/10.1145/1089786.1089794

[5] Juliet M. Corbin and Anselm L. Strauss. 2008. *Basics of qualitative research: techniques and procedures for developing grounded theory*. SAGE Publications, Inc.

[6] Lyn Corno. 1986. The metacognitive control components of self-regulated learning. *Contemporary educational psychology* 11, 4 (1986), 333–346.

[7] Holger Danielsiek, Jan Vahrenhold, Peter Hubwieser, Johannes Krugel, Johannes Magenheim, Laura Ohrndorf, Daniel Ossenschmidt, and Niclas Schaper. 2017. Undergraduate teaching assistants in computer science: Teaching-related beliefs, tasks, and competences. In *2017 IEEE Global Engineering Education Conference (EDUCON)*. 718–725. https://doi.org/10.1109/EDUCON.2017.7942927

[8] Adrian Devey and Angela Carbone. 2011. Helping First Year Novice Programming Students PASS. In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114* (Perth, Australia) *(ACE '11)*. Australian Computer Society, Inc., AUS, 135–144.

[9] Francisco J. Estrada and Anya Tafliovich. 2017. Bridging the Gap Between Desired and Actual Qualifications of Teaching Assistants: An Experience Report. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITiCSE '17)*. Association for Computing Machinery, New York, NY, USA, 134–139. https://doi.org/10.1145/3059009.3059023

[10] Paul Golding, Lisa Facey-Shaw, and Vanesa Tennant. 2006. Effects of Peer Tutoring, Attitude and Personality on Academic Performance of First Year Introductory Programming Students. In *Proceedings. Frontiers in Education. 36th Annual Conference*. 7–12.

[11] Charles R. Greenwood, J.J. Carta, and D. Kamps. 1990. Teacher-Mediated Versus Peer-Mediated Instruction: A Review of Educational Advantages and Disadvantages. In *Children Helping Children*. John Wiley and Sons, New York, 177–205.

[12] Torey Halsey. 2018. *Being an Undergraduate Teaching Assistant at Kalamazoo College: Collaboration and Professional Development.*

[13] Ken Hartness and Li-Jen Shannon. 2011. Peer mentors and their impact for beginning programmers. *Information Systems Education Journal* (01 2011).

[14] Robert J Havighurst. 1953. Human development and education. (1953).

[15] Andrew Hunt and David Thomas. 2000. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Longman Publishing Co., Inc., USA.

[16] Päivi Kinnunen and Beth Simon. 2010. Experiencing Programming Assignments in CS1: The Emotional Toll. In *Proceedings of the Sixth International Workshop on Computing Education Research* (Aarhus, Denmark) *(ICER '10)*. Association for Computing Machinery, New York, NY, USA, 77–86. https://doi.org/10.1145/1839594.1839609

[17] Päivi Kinnunen and Beth Simon. 2011. CS Majors' Self-Efficacy Perceptions in CS1: Results in Light of Social Cognitive Theory. In *Proceedings of the Seventh International Workshop on Computing Education Research* (Providence, Rhode Island, USA) *(ICER '11)*. Association for Computing Machinery, New York, NY, USA, 19–26. https://doi.org/10.1145/2016911.2016917

[18] Päivi Kinnunen and Beth Simon. 2012. My program is ok – am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education* 22, 1 (2012), 1–28. https://doi.org/10.1080/08993408.2012.655091 arXiv:https://doi.org/10.1080/08993408.2012.655091

[19] Paul A. Kirschner, John Sweller, and Richard E. Clark. 2006. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist* 41, 2 (2006), 75–86. https://doi.org/10.1207/s15326985ep4102_1 arXiv:https://doi.org/10.1207/s15326985ep4102$_1$

[20] Dastyni Loksa and Amy J. Ko. 2016. The Role of Self-Regulation in Programming Problem Solving Process and Success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 83–91. https://doi.org/10.1145/2960310.2960334

[21] Dastyni Loksa, Amy J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 1449–1461. https://doi.org/10.1145/2858036.2858252

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

[23] Keith Nolan and Susan Bergin. 2016. The Role of Anxiety When Learning to Program: A Systematic Review of the Literature. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '16)*. Association for Computing Machinery, New York, NY, USA, 61–70. https://doi.org/10.1145/2999541.2999557

[24] Claudia Ott, Anthony Robins, Patricia Haden, and Kerry Shephard. 2015. Illustrating performance indicators and course characteristics to support students' self-regulated learning in CS1. *Computer Science Education* 25 (04 2015), 174–198. https://doi.org/10.1080/08993408.2015.1033129

[25] 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* (Auckland, New Zealand) *(ICER '12)*. Association for Computing Machinery, New York, NY, USA, 11–16. https://doi.org/10.1145/2361276.2361280

[26] Paul R. Pintrich. 2000. The role of goal orientation in self-regulated learning. In *Handbook of self-regulation*. Elsevier, 451–502.

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

[28] Thomas W. Price, Zhongxiu Liu, Veronica Cateté, and Tiffany Barnes. 2017. Factors Influencing Students' Help-Seeking Behavior While Programming with Human and Computer Tutors. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 127–135. https://doi.org/10.1145/3105726.3106179

[29] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017), 24 pages. https://doi.org/10.1145/3077618

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

[31] Emma Riese. 2018. Teaching Assistants' Experiences of Lab Sessions in Introductory Computer Science Courses. In *2018 IEEE Frontiers in Education Conference (FIE)*. 1–5. https://doi.org/10.1109/FIE.2018.8659243

[32] Martin Ukrop, Valdemar Švábenský, and Jan Nehyba. 2019. Reflective Diary for Professional Development of Novice Teachers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 1088–1094. https://doi.org/10.1145/3287324.3287448

[33] Andries van Dam. 2018. Reflections on an Introductory CS Course, CS15, at Brown University. *ACM Inroads* 9, 4 (Nov. 2018), 58–62. https://doi.org/10.1145/3284639

[34] Joseph Weizenbaum. 1966. ELIZA—a Computer Program for the Study of Natural Language Communication Between Man and Machine. *Commun. ACM* 9, 1 (Jan. 1966), 36–45. https://doi.org/10.1145/365153.365168