Reading Between the Lines: Student Help-Seeking for (Un)Specified Behaviors

JOHN WRENN, Brown University, USA
SHRIRAM KRISHNAMURTHI, Brown University, USA

A thorough understanding of specified behavior is essential for the completion of most programming tasks. Researchers have created automated tools to help students with this task. Yet, even with automated feedback, students may still face self-insurmountable challenges for which they must seek aid from the course staff.

What self-insurmountable challenges do students face? And (how) does access to automatic, on-demand feedback shape student help-seeking? To find out, we manually reviewed the 1,247 assignment-related student posts in the online help-forum of a post-secondary accelerated introductory computer science course. We report on the high-level relationships between student help-seeking and (under)specification in assignments, and identify a number of behaviors relevant to both researchers and educators.

CCS Concepts: • **Social and professional topics → Computing education**.

Additional Key Words and Phrases: underspecification, help forum, problem understanding

ACM Reference Format:
John Wrenn and Shriram Krishnamurthi. 2021. Reading Between the Lines: Student Help-Seeking for (Un)Specified Behaviors. In *21st Koli Calling International Conference on Computing Education Research (Koli Calling '21), November 18–21, 2021, Joensuu, Finland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3488042.3488072

1 INTRODUCTION

Students must understand programming problems as well as possible before starting to code. Failure to do so leads to many negative consequences. First, they may get a poor grade. Second, they may experience unnecessary frustration. Third, they may not meet the learning objective of that problem. A poor understanding of a problem can also lead students to pick a poor solution strategy, in which they may get entrenched even if it is inappropriate, trying (often unsuccessfully) to tweak it rather than start afresh [5, 8, 11].

These concerns are not hypothetical. A growing body of literature (see section 2) shows that students frequently misunderstand the problem they have been asked to solve. Therefore, it is important to get students to establish understanding before implementation.

How do we get students to understand problems before starting to write solutions? As we discuss in section 2, researchers have adopted a variety of strategies for this purpose. Several of these amount to having students write *examples of expected behavior* (e.g., in the form of unit tests) before starting to code. Research also shows [2, 7] that these strategies are quite successful.

However, example-writing only works when the problem is fully specified; but unspecified behavior abounds. Sometimes it is avoidable at a cost: e.g., an instructor may choose to elide pedantic details for brevity (such as the

Manuscript submitted to ACM

handling of `null` or extreme numerical inputs). There are also scenarios in which unspecified behavior is all-but-*unavoidable*. If an assignment involves floating point arithmetic, slight variations in arithmetic order may produce slightly different results. Moreover, unspecified behavior is inherent in some algorithms. A `sortByKey` procedure backed by an unstable sorting algorithm will leave unspecified the relative ordering of elements with equal keys. Likewise, a graph can have multiple shortest paths or minimum spanning trees. All these complicate the idea of using input-output examples to confirm student understanding.

In this paper, we draw on Examplar [12], with which students can both confirm their understanding of the problem with tests and be forewarned if they test incorrect or unspecified behavior. In Examplar, the course staff write a collection of known-correct ("wheat") and known-incorrect ("chaff") implementations for each problem. Students' unit tests are run against wheats and chaffs. Correct unit tests should pass the wheats, and a sufficiently rich family of unit tests should correctly identify the chaffs as incorrect. Examplar is provided as an interactive tool built into the student's development environment, and students use it voluntarily [13].

We present Examplar to our students as a kind of teaching assistant (TA). It has a very limited interface: it can only answer questions about the input–output behavior described by the problem specification, and it only expresses its answer in terms of wheat-passing and chaff-catching. However, it is always present, responds immediately, is consistent, and is infallible (we accept any student work that passes it). Therefore, when students have questions about the problem specification, we urge them to "ask Examplar first". Given this infallible oracle of (un)specified behavior, we are interested in learning: (1) *What questions do students still have left about (un)specified behavior?* (2) *What is the influence of automatic, on-demand feedback about (un)specified behavior in these questions?*

2 THEORETICAL BASIS & RELATED WORK

Whalley and Katso [11] identify how students retrieved the wrong schema and "did not recognize that their program would not work and did not attempt to verify the correctness of their solutions". Loksa and Ko [5] found that students "often began coding without fully understanding the problem, leaving them with knowledge gaps in the problem requirements". Prather, et al. [8] observe that the "most frequent issue these students encountered was a failure to build a correct conceptual model of the problem".

Consequently, several authors have tried to build tool support to help students with this step. Prather, Denny and colleagues present students with instructor-chosen inputs and ask students to enter the corresponding outputs [2, 7], much like a quiz. In contrast, we adopt the line of work by Wrenn, et al. [12, 13], who follow the Design Recipe [3] in asking students to write input–output examples but embed it into the development environment.

Given the importance of problem understanding, this paper examines situations where Examplar-style automation *is not enough*. Our closest related work is other projects that have examined student help-seeking. While student help-seeking is a longstanding area of inquiry in education research [4], research into computing students' help-seeking on course forums is comparatively nascent.

Vellukunnel, et al. [10] evaluated forum posts from 395 students enrolled in CS2 courses across two universities. The researchers manually labeled work-related posts by quality and content, and found that some categories positively correlated with grades. However, they do not offer detailed insight into the nature of the content-clarification posts, which is our sole focus here.

We focus on forum posts but ignore help sought in TA hours. We chose to not monitor these hours because students might find that invasive and become reluctant to seek help. Ren et al. [9] studied TA use through a non-intrusive but

lightweight mechanism. The trade-off is that their instrument cannot provide more information into our question, and they anyway find that relatively few questions about input–output behavior are asked during hours.

3 PEDAGOGIC CONTEXT

CSA is an accelerated introductory computer science course at a private university in the USA. Students must place into the course through (self-contained) programming tasks in the summer. Though some students have no prior programming experience at all, most have at least a little, and many have completed Advanced Placement or the equivalent. Its Fall 2020 enrollment was 114 students, of whom 92 were first-years and about 20% female. CSA covered rich data structures, graph algorithms, lazy streams, and Big-O.

Assignments. CSA had 14 relevant programming projects (and no exams). For all assignments, students were provided a handout describing the input–output behavior of the functions to implement, and were expected to submit a code file with their solution. For most, they were provided template files with stubs of the functions to implement. Most assignments also expected them to submit a separate file containing their test suite.

Grading. CSA's manual grading did some correctness checking, but focused much more on design and stylistic issues (and also evaluated non-code aspects like Big-O analyses). Each of the code and test files were separately auto-graded. The code was graded against a staff-authored test suite. This feedback was withheld from students until after the assignment deadline.

Students tests were graded against wheats and chaffs (described in section 1). Students could run Examplar at any time before the deadline; these were not monitored or graded. Because students can spend too long catching chaffs by writing ever-more-complex tests [12], CSA included only a small number (4–6) of chaffs in Examplar. This limits the amount of time they spend on *problem understanding*. However, they are expected to write much more thorough test suites to *identify bugs* in their implementations. During grading, their test suites were run against many more chaffs (upwards of 20). Their testing grade was based on how well their valid tests did against these chaffs. Unlike with code, they could obtain testing feedback against the full panel of wheats and chaffs at any time.

Student Help Resources. The Fall 2020 offering had 15 undergraduate TAs who cumulatively provided 30 hours per week of one-on-one live assistance to students, and answered questions posted to an online help forum (as did the professor). Questions posed to the help forum were, as a rule, only visible to course staff; students could not see the questions posed by other students. When, occasionally, course staff changed the visibility of a students' post to 'public', the poster's identity remained hidden to other students.

Impact of COVID-19. Owing to the pandemic, the course was entirely virtual. Enrollment was about twice normal but with only a 25% increase in course staff. Many students took this as their only course. Normally, students meet with TAs in person and value the interaction (often waiting in long lines to do so). However, TA sessions were also made virtual (and staggered to handle time zones). For whatever reason, students posted many more forum questions (with the same software, Campuswire) than before: 1,602 posts, which (after accounting for class size difference) is 54% more than in Fall 2019, for largely the same assignments.

4 NAVIGATING (UN)SPECIFIED BEHAVIOR

Automated grading demands that students accurately match the behavior described by each assignment's specification. To succeed, students needed to precisely interpret both the specified *and* unspecified behaviors of each assignment. As

many of our findings (section 6) relate to how students responded to (un)specified behaviors, we briefly review the challenges they pose in this section.

Unspecified Behavior. The *unspecified behaviors* are those aspects for which the programmer has discretion over the exact behavior of their program. For example, what is the mean of an empty list? Error? Exception? 0?

Resolving Invalidity. On such behaviors, a concrete test written by a student may clash with Examplar's wheat, making their test invalid. What a student does next is complicated. Perhaps they should write more tests to understand the problem better. Maybe they have an errant test and should remove it. Or, most subtly, it may be that the test involves *semi-specified* behavior, in which case the student should use *property-based testing* (PBT) [1]. For instance, a thorough test suite of the mode *should* include inputs with multiple modes, but it should not assert that the output for such inputs is any *one* value. Rather, it must check that the output satisfies a property: that it is *a* mode of the input list.

Pedagogy. To prepare students to handle semi-specified behavior, we devote two assignments entirely to PBT. For these assignments, students' sole objective was to produce a *testing oracle*, a function that consumes a purported implementation, generates and runs tests, and labels it as (potentially) correct or (known) buggy.

We reference the first of these, SORTACLE, below. In it, students implemented a predicate that consumes two lists of `Persons` (records with a `name` and an `age`) and checks if the second list is an instance of the first sorted by `age`. Here, the *precise* ordering of `Persons` with the same age is intentionally left unspecified.

5 METHODS

Of 1,602 forum postings overall, we harvested the 1,247 that were tagged with an assignment. We manually filtered for those relating to *input–output* behavior of assignments. This netted 298 postings.

We reviewed these both per-assignment and cross-assignment for common (or otherwise notable) phenomena. We additionally attempted to classify whether each posting was prompted by Examplar, and whether it related to unspecified inputs, semi-specified outputs, or the specified behavior of the assignment. We considered a posting to be prompted by Examplar if the student mentioned Examplar explicitly, included a screenshot of its feedback, or alluded to its feedback (e.g., "Why is this test invalid?"). A posting related to unspecified inputs if it concerned an input outside the domain of the assignment; it related to semi-specified outputs if it concerned an input admitting multiple possible outputs; and it related to the specified behavior of the problem if it concerned a case in which the input is specified and admits one valid output. We left particularly hard-to-classify postings—nine, in all—unclassified.

The first author did all the classification (so inter-rater reliability does not apply). For hard-to-classify postings, this author sought feedback from former course TAs. The second author (who was the course instructor) provided context and interpretation for the first author's findings. As this interpretation is innately subjective, we center our presentation on the quoted postings of students.

We pay special attention to the ninth assignment, TWEESEARCH. It asks students to implement a search function that consumes a list of "tweets" and a query to fuzzily search for, and produces a list of tweets sorted by relevance. The *exact* output order of equally-relevant tweets is unspecified, so there can be several valid answers. Examplar will reject as invalid any test that asserts an exact order. Students were told to treat this like an exam: "without consulting course staff except for critical issues (broken links, possible assignment typo, etc.)". Therefore, any input–output behavior questions about this assignment would have been posted as a last resort.

6 OBSERVATIONS

We now try to answer our research questions. We begin with basic statistics about the kinds of questions asked, then move on to several observations that we believe may be insightful to researchers and educators. The 298 postings we reviewed were authored by 90 different students. The most prolific of these students authored 12 input–output related posts; the median student authored three.

6.1 Distribution of Questions

We begin by examining the distribution of the 230 input–output related questions students posed across 12 programming assignments (we exclude the 68 postings relating to Sortacle and Oracle, since the ed unusual nature of these assignments precluded providing validity feedback) across several basic axes.

Specified or Unspecified? There is a big difference between (what a student believes to be) specified versus unspecified behavior. If the behavior appears to be specified, a student is more likely to ask why the outcome is or is not what it is. If it seems unspecified, the student does not know what should happen at all. We found 106 to be about specified behavior and 110 about unspecified; a few could not be definitively classified, or included questions of both kinds.

Kinds of Unspecified Behavior. On most assignments, students grappled with one of two kinds of unspecified behavior: inputs for which no output was specified, and inputs for which the output was only semi-specified. Among the 110 postings that included questions about unspecified behavior, 58 concerned unspecified inputs, and 60 concerned semi-specified outputs.

Relationship to Automated Feedback. Within the ten assignments for which students had automated feedback about the validity of their tests, students posed 169 input–output-related questions; of these 105 mentioned automated feedback that a test was invalid. Of these 105 questions, 28 concerned specified behavior, 29 concerned unspecified inputs, and 43 concerned semi-specified outputs.

6.2 Not Using Examplar's Feedback

Despite the availability of Examplar, students posed questions that could have been answered automatically with a test case; e.g.: «*Do the movement functions in Updater need to be repeatable/stackable?*» For such questions, we might attribute help-seeking on the course forum instead of Examplar to a preference for prose. One student, asked why they had not answered their question using Examplar, clarified that they had been pen-and-papering out examples and had not yet realized validity feedback was available.

Other cases are more puzzling. In several instances, students asked the TAs whether a *particular* test case was valid or not:

> *«Should we be allowed to update multiple nodes or navigate around the tree without turning the cursor back into a tree between updates. For example, would a test like this be consistent with the problem specification? [test case code elided]»*

This question could have been answered by Examplar, but the student does not mention having tried it. Is it possible that they did not know Examplar's validity feedback corresponds to "consistent with the problem specification"? Or do they prefer to ask staff instead of tools? For similar examples, see section 6.5.

6.3 Input Bias

When faced with feedback that a test was invalid, students needed to determine whether the invalidity was caused by an unspecified input, or an over-constrained assertion on semi-specified output. Multiple students *mis-attributed* the invalidity of their over-constrained tests to unspecified inputs.

Recall that in TweeSearch, testing for a *particular* output order of equally-relevant search results is invalid. However, at least six students misattributed the cause; e.g.:

> *«[C]an we assume that two tweets in our list will never have the same overlap with a search tweet? I think the answer is yes since Examplar errors no matter the order of the outputted list, but I was unsure because it doesn't error if you check the length of the outputted list.»*

This student misattributed their test invalidity to the input, and thus asks if they can (incorrectly) assume that inputs that admit equally-relevant results are invalid.

Such misattributions are worrying. They might push students to delete their over-constrained tests, rather than to reformulate them to be property-based. Much worse, students may end up implementing an incorrect solution that is brittle in the face of valid inputs that they have incorrectly ruled out.

This posting is notable in that the student correctly intuits the method for distinguishing between invalid inputs and semi-specified outputs: use PBT. Other students were not so fortunate, and even this student only presents this as a passing observation, not recognizing that this is in fact what they are expected to do. This, at least, suggests a pedagogic flaw in the class.

6.4 Failure to Transfer

Even though students had completed two assignments devoted to PBT, on postings for later assignments with semi-specified outputs, course staff frequently had to prompt students to *remember how they had handled this situation in the past*. This happened both soon after those assignments, and late in the semester.

On TweeSearch, which came two weeks after the latter PBT assignment, students needed to adapt the PBT they developed in Sortacle. Unfortunately, 23 of the 36 input–output-related posts on this assignment concerned testing sortedness. Several students drew a partial connection to PBT, but struggled to find a middle-ground between exact-value testing and PBT; for instance:

> *«If we have tweets that have the same overlap coefficient w/ respect to the new tweet, should our tests allow for any ordering of these overlap coefficients, or is there a secondary characteristic of the tweets that we should be sorting by […]? Based on what Examplar seems to say, these tests are not valid at all if I'm checking order. However, if I just check whether the resulting lists have the same elements in them, Examplar returns the expected result. Although this means of testing works, this is not necessarily ideal considering it is not necessarily true that order is ENTIRELY irrelevant.»*

This student has clearly read the specification closely, identified that exact-value testing is inappropriate, and succeeded with weak PBT—but they were unable to commit to this strategy. (See also section 6.3.) These failures suggest difficulties that students face with testing against properties rather than concrete outputs.

6.5 Specification Preconceptions

Postings regarding the *specified* behavior of functions often suggested the student carried preconceptions about the intended behavior of the assignment.

Imagined Functionality. Students wondered whether they needed to provide functionality not mentioned by the assignment specification. For instance, on a recommendation system for books, two students wondered about case-sensitivity; e.g.: «*When matching titles, do we assume they match exactly, or should we take into account capitalization? Should we also take other words into account?*» The assignment handout does not suggest that the recommender should do anything other than check exact matches, but such functionality *would* be useful in a real-world version of the assignment.

Imagined Constraints. On Sortacle, in addition to building the checker, students had to read and reflect on the article "Falsehoods Programmers Believe About Names" [6]. Nonetheless, among the 43 input–output-related questions posed by students, over a dozen concerned the well-formedness of names and ages; e.g.: «*Can the name of a person be literally any String? It's not specified in the assignment. For instance, can Strings like "@#$&@^% 1112" or "Hello World" or even "X ÆA-12" constitute names?*» All of these were effectively answered by the handout (which does not rule them out, and references the article, making these invalid constraints).

This tendency extended, to a lesser extent, to more abstract assignments. On Updater, two groups of students wondered about validity constraints on trees; e.g., one wrote: «*For a single node, is it guaranteed that its children will have distinct values?*» There was nothing in the specification suggesting that the data carried by sibling nodes ought to be unique.

6.6 Expanding the Specification

Whether through prior conditioning or other reasons, many students seem to not entirely understand, or perhaps even believe, that the assignment constitutes a kind of contract, and that the course staff will not demand, and often do not even care about, aspects not specified. For instance, on a problem to compute text overlap, the function was only specified on non-empty lists (since the formula does not make sense otherwise). Nonetheless, students wanted to know the "right" thing to do for empty inputs.

Theoretically, unspecified inputs ought to reduce work for students: their use helps keep specifications shorter, and they reduce the surface area of functionality that students must implement and test. In practice, however, these postings suggest that unspecified inputs may be a source of anxiety, at least early in the semester.

6.7 Not Understanding Examplar

Some students' postings indicated that they did not have a clear execution model for the validity feedback produced by Examplar, even late into the semester. We saw three main categories of confusion.

What is Checked? As an example, on ContinuedFractions, the eighth programming project of the course, one student asked:

> *«I tried testing threshold as followed below but received "These tests do not match the behavior described by the assignment" [...] I don't know if it is my implementation or something to do with an invalid test.»*

Feedback that *tests* are invalid can only indicate an issue with tests. Because the student's tests are run against instructor implementations, and the tests can be run before the student's implementation has even begun, invalidity cannot *possibly* be a statement about the student's implementation. We unfortunately see similar misunderstandings late into the semester too.

Mis-Experiments. Of the 23 postings to TweeSearch about its semi-specified behavior, 16 explained that they had individually tested *every* possible ordering—all of which were (correctly) rejected as invalid by Examplar. Reactions included bug reports; e.g.:

> *«I think there's something broken. The following checks all fail, which is interesting because they seem to cover every single possible output. Am I missing something, or is the [wheat] broken?»*

Reifying the Model. An earlier version of Examplar [12] reported the number of wheats it uses to check validity. The tool's designers "simplified" its interface by removing this information [13]. Sadly, several postings suggest that hiding this detail of the execution model may have been counterproductive; e.g.:

> *«If our tests are only being tested against one wheat, then something weird is happening. If there are deliberately multiple wheats to ensure that our tests are order-agnostic, then it'd be good to know.»*

This student had formed a perfect conception of the situation and, had they been given the wheat count, they would have come to exactly the right understanding without needing to ask.

7 LIMITATIONS

We do not know what questions students asked course staff in office hours. • Our classifications of postings was done by just one person. • Our interpretations are best-guesses, informed by our experience with the course, and need to be validated by future interventions and research. • Different student populations, in different conditions, would likely encounter different challenges. • Other instructional staff would vary in what they anticipated or failed to anticipate. • This work only examines problems that, while sophisticated, for the most part: do not involve state or other side-effects; have agreed-upon data structures; and have a clear goal. Removing each of these makes "automated TAing" much harder.

8 DISCUSSION

Our observations confirm that Examplar can serve as a guardrail for students, alerting them to their problem misconceptions *before* final grading [12]. Many of these instances we cannot—by definition—see in our current dataset, but the 105 postings that cited Examplar's feedback reflect a lower-bound for the number of instances where a student was stopped from veering off-track.

However, postings also suggest that many students did not know how to respond to feedback about invalidity. We observe a time-consuming anti-pattern too, reminiscent of Prather et al.'s [8] observations of students consumed by automated feedback on their implementations: students reacting to invalidity by exhaustively testing every possible output of semi-specified functions. These students were not adequately prepared for testing unspecified behavior the course's PBT-focused assignments. And, at least one student who *did* understand Examplar's assessment model for unspecified behavior was hampered by the tool's opaque feedback («*If there are deliberately multiple wheats [...] it'd be good to know.*»). Educators must address how to help students respond productively to invalidity, especially if caused by unspecified behavior.

More broadly, *problem* comprehension seems to demand new skills that are not very well covered in current curricula or standards. For instance, early examples and subsequent testing both require good *adversarial* thinking: a mindset that is not usually included in definitions of "computational" thinking but is clearly relevant even from an early stage (not only in areas like security). Yet, the barriers to adversarial thinking may be more social than instructional or technical:

the availability of Examplar did not stop students from posing questions about the expected input–output behavior of problems. Future work should consider the possibility that students prefer the assurances of humans over that of automated feedback.

ACKNOWLEDGMENTS

We thank Campuswire for providing a forum for our students. We appreciate the patience of our students and apologize to them for the shortcomings that this paper lays bare. This work was partially supported by the US NSF.

REFERENCES

[1] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*. Association for Computing Machinery, New York, NY, USA, 268–279. https://doi.org/10.1145/351240.351266

[2] Paul Denny, James Prather, Brett A. Becker, Zachary Albrecht, Dastyni Loksa, and Raymond Pettit. 2019. A Closer Look at Metacognitive Scaffolding: Solving Test Cases Before Programming. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '19)*. Association for Computing Machinery, New York, NY, USA, Article 11, 10 pages. https://doi.org/10.1145/3364510.3366170

[3] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2001. *How to Design Programs* (first ed.). MIT Press, Cambridge, MA, USA. http://www.htdp.org/

[4] Sharon Nelson-Le Gall. 1985. Chapter 2: Help-Seeking Behavior in Learning. *Review of Research in Education* 12, 1 (1985), 55–90. https://doi.org/10.3102/0091732X012001055

[5] Dastyni Loksa and Andrew 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)*. ACM, New York, NY, USA, 83–91. https://doi.org/10.1145/2960310.2960334

[6] Patrick McKenzie. 2010. Falsehoods Programmers Believe About Names. https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/

[7] James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. 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, 531–537. https://doi.org/10.1145/3287324.3287374

[8] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/3230977.3230981

[9] 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

[10] Mickey Vellukunnel, Philip Buffum, Kristy Elizabeth Boyer, Jeffrey Forbes, Sarah Heckman, and Ketan Mayer-Patel. 2017. Deconstructing the Discussion Forum: Student Questions and Computer Science Learning. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 603–608. https://doi.org/10.1145/3017680.3017745

[11] Jacqueline Whalley and Nadia Kasto. 2014. A Qualitative Think-aloud Study of Novice Programmers' Code Writing Strategies. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (Uppsala, Sweden) *(ITiCSE '14)*. ACM, New York, NY, USA, 279–284. https://doi.org/10.1145/2591708.2591762

[12] John Wrenn and Shriram Krishnamurthi. 2019. Executable Examples for Programming Problem Comprehension. 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, 131–139. https://doi.org/10.1145/3291279.3339416

[13] John Wrenn and Shriram Krishnamurthi. 2020. Will Students Write Tests Early Without Coercion?. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '19)*. Association for Computing Machinery, New York, NY, USA, Article 27, 5 pages. https://doi.org/10.1145/3428029.3428060