How should we 'Explain in plain English'? Voices from the Community

Max Fowler
mfowler5@illinois.edu
University of Illinois
Urbana, Illinois, USA

Binglin Chen
chen386@illinois.edu
University of Illinois
Urbana, Illinois, USA

Craig Zilles
zilles@illinois.edu
University of Illinois
Urbana, Illinois, USA

ABSTRACT

"Explain in plain English" (EipE) questions are seen as an important developmental activity and assessment tool in the research community studying how people learn to program, but they aren't widely used in practice because of difficulty of grading and workload issues. In this paper, we interviewed eleven members of the introductory programming education research community about their thoughts on EipE questions as a whole and how individual borderline student answers should be graded. Through inductive coding of the interview transcripts, we identify: (1) themes relating to how EipE questions should be used in class, (2) the importance of training students to complete EipE questions, (3) standards for the selection and presentation of code in EipE questions, (4) the theoretical and practical considerations relating to grading EipE questions, and (5) English as a second language (ESL) concerns. In addition, we attempt to extrapolate from our observations what the underlying grading process is that faculty are using to grade EipE questions.

CCS CONCEPTS

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

KEYWORDS

interviews, explain in plain English, EipE, qualitative

ACM Reference Format:
Max Fowler, Binglin Chen, and Craig Zilles. 2021. How should we 'Explain in plain English'? Voices from the Community. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021), August 16–19, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3446871.3469738

1 INTRODUCTION

"Explain in plain English" (EipE) questions, as shown in Figure 1, provide a student with a code snippet to read and ask the student to describe in natural language what it is that the code does. Students are generally directed to not provide a line-by-line description, but rather a holistic description that demonstrates that the student understands the interplay between the lines of code.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICER 2021, August 16–19, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8326-4/21/08…$15.00
https://doi.org/10.1145/3446871.3469738

For each of these sections of code, explain in plain English what it does. For example, "It displays the sum of the two integers" or "It returns the last value in the array." Don't describe the code line-by-line; state what it does overall.

i. (Nested ifs) Assume a, b and c are all int variables that have been declared and initialized.

```
if ( a < b)
   if ( b < c)
       System.out.println( c );
   else
       System.out.println( b );
else if ( a < c)
       System.out.println( c );
else
     System.out.println( a );
```

Figure 1: Example 'Explain in plain English' question from Murphy *et al.*[21].

EipE questions were popularized in the mid-2000's as a means of assessing students' ability to read code, a developmental skill related to learning to program [14]. Researchers theorized that there is a loose hierarchy of programming skills with code writing at the top of the hierarchy and many programming students struggling with tasks lower in the hierarchy [15]. These skills span from understanding syntax (as the easiest), to code tracing (executing code in your head for one particular input), to code reading/explaining (abstracting the behavior of code across all possible inputs), to code writing (as the most complex) [15]. We discuss this prior work in Section 2.

In spite of likely playing an important role in learning to program, code reading/explaining activities like EipE questions aren't heavily used in introductory programming classes because of the effort and difficulty of grading them (see Section 4.2). EipE questions are somewhat unique among CS 1 assessment items, because they involve natural language responses. Answers to syntax, tracing, and code writing questions can be cleanly delineated into correct and incorrect answers, making them easier to grade and frequently allowing them to be graded automatically in large CS 1 courses. The lack of an absolute line for objective correctness makes EipE grading far less straightforward.

Nevertheless, natural language processing (NLP) has been proposed as a means to automate the grading of EipE questions [2, 10] for large classes. However, such a proposal immediately arrives at the question, "But, which answers should the algorithm mark as correct?" While we endeavored in prior work to establish a rubric for EipE [5], our experience using EipE produced myriad questions and difficult scoring decisions. In this work, we sought out to explore if

there was a community consensus on how EipE questions should be graded. Specifically, we considered two research questions:

(1) How should EipE questions be scored?
(2) What are best practices around EipE questions (e.g., how they should be used, presented, etc.)?

To explore these research questions, we conducted structured interviews with eleven members of the introductory programming education research community. The questions on these interviews and a description of our inductive coding of these interviews is presented in Section 3. To our knowledge, this is the first attempt to synthesize community knowledge around EipE question.

We find a lot of agreement in the goals for, construction of, presentation of, and student preparation for EipE questions (Sections 4 and 5). On the surface, there was less consensus about exactly how scores should be assigned to answers (Section 6), but we believe there are broad similarities about how our participants assigned grades (Section 7). We are cautiously optimistic that a community standard for grading EipE questions might be achievable.

2 RELATED WORK

EipE questions were popularized in the mid-2000's, appearing in a study by Whalley et al. in which students were asked to, "In plain English, explain what the following segment of Java code does" [29]. These questions are a way of exploring students' ability to grasp the abstract meaning of code segments, rather than just be limited to explaining the behavior of individual lines of code. Answers to Whalley's EipE question were classified using the Structure of Observed Learning Outcome (SOLO taxonomy) [4]. SOLO is frequently used in existing EipE literature as a metric for categorizing EipE answers [9, 21]. While the full taxonomy features five categories, the two most relevant categories to code EipE are multi-structural – line-by-line descriptions of a code snippet – and relational – abstractions of what a code's purpose is. Since the Whalley et al. study, several authors have investigated the relationship between code reading and code writing [16–18, 21, 22].

To provide an example of what multi-structural and relational answers for EipE prompts are, we use Figure 1 from Murphy *et al.* and their study linking EipE proficiency to code writing on a computer. This Java code snippet has three int variables and a decision tree. A relational explanation for this code snippet would be something along the lines of *given three ints, print the largest of the three.* This answer is relational because it correctly summarizes the overall purpose of the code snippet. By contrast, we can construct a correct but multi-structural answer as follows: *This code has int a, b, and c. If a is less than b and b is less than c, the code prints c. If a is less than b and b is not less than c, the code prints b. If a is less than b and less than c, the code prints c. If none of these are true, the code prints a.* This is a correct written tracing of the decision tree's structure, but the overall purpose of the code is absent.

Whalley et al. argue that comprehension of a piece of code, and understanding of the knowledge used within it, are necessary prerequisites for novices to write that same code. Specifically, a *relational* level of understanding and not just *multi-structural* understanding is necessary for programmers [29]. Longitudinal studies have shown that students who cannot explain code relationally early in the semester tend to struggle with writing code later on [8].

Further work in this area provides evidence for a hierarchy of programming skills, with mastery at lower levels of the hierarchy shown to be predictive of code writing ability [8, 17, 18, 27]. In particular, Lopez et al. find that students' performance on tracing and code reading questions account for 46% of the variance in their performance on code writing questions [18]. Although these studies do not state the hierarchy has a strict structure, Lister et al. state that, "We found that students who cannot trace code usually cannot explain code, and also that students who tend to perform reasonably well at code writing tasks have also usually acquired the ability to both trace code and explain code." [17]

Multiple authors propose that code tracing and reading should be a greater focus in novice instruction [1, 6, 7, 17, 22, 30]. Lister et al. state, "It is our view that novices only begin to improve their code writing ability via extensive practice in code writing when their tracing and explaining skills are strong enough to support a systematic approach to code writing ..." [17]

Code reading activities may be among the best activities for instructing novices in the use of common programming patterns. One key way experts differ from novices is their ability to automatically 'chunk' multiple syntax elements and process them as one unit [6, 11, 20, 31], reducing their cognitive load [25]. These chunks (or *schema* in the cognitive load literature) are iteratively constructed through repeated use of and exposure to common identifiable features [19] and are learned more efficiently in lower cognitive load activities (i.e., code reading rather than writing) [26]. When students had learned an applicable pattern, Rist found that students could and did reason forward from plan to code [24].

3 METHOD

We contacted 19 members of the community by email to participate in interviews. Interview subjects were drawn from authors of papers involving EipE questions and other prominent members of the research community studying introductory programming that were familiar with or who had used EipE questions in introductory programming courses. Eleven respondents (7 women, 4 men) agreed to participate in interviews (58% response rate), two others indicated that they are no longer engaged in work at all related to CS 1, and six did not respond. Our participants were from five countries on three different continents; they predominantly taught large enrollment courses with the help of teaching assistants, but at least one of them did all of the grading themselves.

We conducted 30 to 60 minute interviews with the participants individually using online video conferencing software. While there was some variation between the interviews due to time constraints, we asked each participant a series of general questions and to comment on a collection of student answers to EipE questions. The general questions are shown in (Figure 2), where the first six were asked at the beginning of the interview and the last two at the end.

Interview subjects were typically asked to consider 16 responses to 5 EipE questions shown in Figure 3. The questions were based on the Python language, and, because not all interviewees were proficient with the Python language, the interviewer ensured that each

(1) Do you use EipE questions in your teaching? Why/why not?
  (a) If used: What was your grading rubric?
(2) What are/would be your goals be in asking EipE questions?
(3) Do you think EipE questions are good for homework, exams, or both? Explain.
(4) What makes a piece of code appropriate for asking on an EipE question?
(5) What aspects are important about how EipE questions are presented to students?
(6) How should functions and variables be named?

(Discussion of specific student answers happens here.)

(7) Is it more important to grade an EipE question based on a student's ability to communicate what the code does or on your perception of the student's understanding of what the code does? Explain.
(8) Do you think there is an EipE grading rubric needs to be specific to the question being asked or is there are universal rubric that works for all questions? Explain.

Figure 2: Participants were asked six general questions before discussing specific student answers and two afterwards.

EipE questions	Interview student answers	Research Motivation

Assume x is a list of numbers (ints and/or floats).
`def f(x):`
`  if len(x) == 0:`
`  return None`
1 `y = 0`
`for i in range(1, len(x)):`
`  if x[i] < x[y]:`
`  y = i`
`return y`
	1a) returns the smallest index in list x	Good key words, wrong relationship

	1b) Initialize y to zero. For each index of the list, check to see if the value at the current index is less than the value at index y. If so, replace y with the current index. Return y.	Canonical multi-structural answer
	1c) Return the index of the minimum number in given list	Correct relational answer, but doesn't consider duplicates in given list

Assume that x is a number
`def f(x):`
2 `if x < 0:`
`  return -x`
`return x`
	2b) return the absolute value of variable x if it is negative	Relational, but incomplete answer

	2c) makes x positive	High-level, but ambiguous
	2e) returns a negative if the number is negative and positive otherwise	Ambiguous about how negation works
	2f) If x is negative, return x as a positive number, then return x as a negative	Incorrect, double return
	2h) If the variable x is less than zero, return that variable negative. Otherwise, just return x	Multi-structural, but on a short piece of code

Assume x is a list of integers and y is an integer
`def f(x, y):`
3 `for val in x:`
`  if val == y:`
`  return True`
`return False`
	3a) Return if y is in list x	Ambiguous meaning of "return if", ask follow-up about "return whether"

Assume x is a list of integers and y is an integer
`def f(x, y):`
`  z = 0`
4 `for val in x:`
`  if val == y:`
`  z += 1`
`return z`
	4a) counts how many times a variable is in a list	Implicitly specifies return and parameters

	4b) counts how many values in x are equal to values in y	How is "values in y" interpreted?
	4c) increases value of z everytime y is in the list	Partly relational, partly multi-structural
	4d) if a value in given list is equal to the value of y, increase z by 1, return z	Incomplete and multi-structural

Assume x and y are both numbers
`def f(x, y):`
5 `if x > y:`
`  return x`
`return y`
	5a) determines if x is greater than y	Incorrect return type/value

	5b) returns the largest number	Unspecified domain (i.e., parameters)
	5c) find out which number is bigger	Ambiguous return type

Figure 3: Participants were shown 5 EipE questions in the Python language: (1) index of min, (2) absolute value, (3) membership, (4) counting value 'y' in list 'x', and (5) max. In each, students were asked to describe the highlighted code and provided the associated type information about the parameters. After understanding each code segment, interview participants were asked to score a series of student answers. The motivation for the selection of each of these answers is included. The gaps in answer numbering resulted from redundant questions used during a test interview that were removed because they were redundant.

participant fully understood the code segment before displaying the student answers. For each student answer, subjects were asked to discuss: (1) what they liked/disliked about the answer, (2) how they would grade it on a binary correct/incorrect scale (and why?), and (3) how they would grade it on a continuous (percentage) scale (and why?). The sample Python code questions and answers were based on answers which we found difficult to score in our own experience, as well as the grading experiences of course teaching assistants [2, 10].

With IRB approval and informed consent, we video-recorded the interviews. The audio portion of each recording was transcribed. The transcripts were inductively coded in groups of 1-3 transcripts at a time, independently by the same three researchers. After each group of 1-3 transcripts, the three researchers met and discussed and refined emergent themes. Having completed per-transcript analysis, the three researchers independently summarized the themes from the coding, and then the group met to arrive at consensus about the themes. These themes are discussed in Sections 4 to 6.

4 RESULTS: WHY AND HOW TO USE EIPE QUESTIONS

4.1 Why should we use EipE questions?

The community members that we spoke to suggested a broad range of motivations for using EipE questions, but the most frequently occurring was that EipE questions involve abstraction. This ability to abstract was seen to aid in debugging, communication, and functional decomposition.

> The most powerful idea in computer science is abstraction, and [EipE] hits it on the head in a way that's very hard to get to.

Others stated that they wanted students to recognize that code has a purpose and to be able to recognize that purpose like experts do.

> We noticed that when [experts] were looking at questions, they didn't immediately trace them, they were looking for the meaning in the code ... if that's the way experts are looking at code, then that's the way we should be teaching students to look for meaning in code.

Additionally, our subjects noted that EipE questions were useful to instructors as a means of assessing what their students know in two forms. First, as a means of identifying what misconceptions the class is holding, and, second, as another instrument (along with tracing, Parson's problems, code writing, etc.) to triangulate on a student's understanding.

4.2 How do we and should we use EipE questions?

Our subjects cited two central concerns that influenced their use of EipE questions: workload and grading difficulty. Many subjects cited that their "classes tended to be very large and there didn't seem to be a way to scale it." Furthermore, a number of subjects cited the difficulty of grading EipE questions, citing "the answers were so messy" and "the difficulty of getting the TAs on the same page." These barriers prevented some of our participants from using EipE questions at all in their courses.

Those subjects that used EipE questions used them in multiple ways. Most subjects reported primarily using EipE questions on summative assessment due to the large amount of time and effort that would be required to grade EipE homework. Those subjects that reported using such questions on homework typically graded them for completion rather than correctness. Some subjects attempted to address the workload issues by casting EipE questions as multiple-choice questions, often selecting distractors from previous student work, but they didn't believe such an implementation to be a real substitute for free response EipE questions.

In spite of current practice, the majority of subjects believed that EipE questions were most useful on formative assessments.

> Formative, but I think my answer to almost any assessment would be: questions are more useful in a formative sense... Summative is for producing numbers at the end.

In addition, a number of faculty indicated EipE questions should be included on both homework and exams if used on either. In addition, they cited the importance of giving feedback.

> When we're doing it for assessment, it's partially because we want the assessments to reflect the kinds of activities that they are doing in class. They've seen these, so we need to ask some of them.

> I don't like to ever test anything that I don't give on homework. So, if I'm gonna have it on an exam, I'm gonna make sure that they can practice with it in homework, so the exam isn't a surprise.

Many faculty cited the utility of EipE questions as in-class activities, typically having students work in pairs or groups to explain the code to each other. For some, this was a way of engaging the students with EipE in a way that didn't require grading, but for others this was part of preparing students to succeed on EipE questions on assessments, which we address in Section 4.3.

4.3 Training students to 'Explain in plain English'

The vast majority of our subjects were adamant about the importance of teaching students how to correctly answer EipE questions.

> I think you just have to put in a lot of training work to make sure that students understand how to answer them in the way that you will grade them as correct.

> You know, you should, in fact, teach students what you want them to do. And, once I did that, they did much better, of course.

Participants noted that students initially struggle with EipE questions both because the task itself is a challenging one and students struggle understanding the difference between a low-level, line-by-line description (referred to as *multi-structural* in the SOLO taxonomy as applied to EipE questions [29]) and a high-level, purpose-oriented description (referred to as *relational* in SOLO). Faculty said it was particularly important to show examples of (incorrect)

multi-structural answers and compare them to (correct) relational answers so that students could learn what constitutes a correct answer.

The training students receive in answering EipE problems was a fundamental component of faculty grading practices. Faculty who had concerns about students' experience with EipE questions tended towards leniency in their grading criteria. In contrast, those that felt that they prepared their students adhered to stricter grading criteria.

> Unless you taught very precisely, used [EipE] questions a lot throughout the semester and raised these sorts of issues with [the students] as you went, I would be not inclined to hit them too hard in the final test.

> If they were to [write a multi-structural answer] in a summative sense, we would say, "Look, you didn't do the thing that we practiced a lot."

In teaching students to solve EipE questions, a number of participants noted that instruction should start with small examples and work up to more complicated ones.

> You have to start with things where the step between what you're asking for and what they can do is pretty small, and then you expand it. It's the "zone of proximal development" stuff again, right? You make it bigger and bigger as the term goes on.

5 RESULTS: CRAFTING AND PRESENTING EIPE QUESTIONS

5.1 What makes a good EipE code segment?

Good EipE code segments are identified by a few common traits. Most broadly, EipE code segments should be code that "has a purpose." This is code where the name for the operation or the overall behavior can be clearly described at a high level. One interviewee identifies this as, "...the code has to be doing something real," and another refers to good EipE code as, "...things that are purposeful as opposed to kinda random."

> What makes a good question later, I think, is one where there is a common structure, a structure that you want them to recognize in a lot of different situations, which has a general purpose...

Many subjects suggested simple code structures that have been previously referred to as introductory patterns [3, 12, 13, 23, 28]. Examples include: "find the biggest thing", "sum up all the elements", "a gathering kind of activity", and checking properties of collections.

One interviewee noted that having loops seemed to be fundamental for a piece of code that was suitable for summarization.

> I found that mine were usually of the form of a loop with an if statement inside... I tended to do things that would either involve arrays or repeated inputs... where people can sort of identify patterns.

Another interviewee emphasized the importance that the behavior of the code should not be guessable from a single line of code but should require the synthesis of ideas from multiple lines of code. Using an example of code that checks for an array being sorted, they said:

> And so a student has to make the inference from looking at each pair that it checks that the whole array is sorted. Those are the ideal questions, where the student has to make the inference that the overall computation does something beyond each piece of it.

Multiple interviewees suggested that there was a sweet spot for size for EipE code segments. A number of subjects asserted that the segments should not be too large or complicated because it would make the problems overwhelming and students might struggle to identify the correct level at which to describe the code.

> So I think that there's one bucket of things that's just, like, too complicated for someone to succinctly describe or like sensibly abstract and describe.

In contrast, for small code segments, many participants were worried that there is no discernible difference between relational and multi-structural answers. One interviewee states that exactly on a short piece of code, "It's hard... There's so little for a student to (pause) Yeah, there's almost no distinction between multi-structural and relational."

With respect to the presentation of these segments for EipE, one interviewee noted that there is no need for the segments to be presented in a specific language. Instead, pseudo code may also be suitable for displaying the code segments used in EipE questions.

Finally, many interviewees indicated that the code segments that are suitable for EipE questions are different from those that are suitable for tracing questions. Code for tracing questions is often designed not to have an easily guessable purpose so that students have to diligently perform bookkeeping in order to compute the code's output. While tracing questions are sometimes designed to be tricky, "[EipE] questions are not supposed to be about trickery."

> That's not a good example for an EipE, because [a tracing question] is not really doing anything, it's just manipulating values within an array for no specific purpose other than to tell whether or not they understand how to do nested indexes.

5.2 Tension in choosing names for functions and variables

Almost universally, our subjects noted a tension when discussing how EipE questions should be presented with respect to variable and function names. This tension revolved around the dual desires of modeling good style to students and not wanting students to be able to guess answers from variable names alone. When asked how variables and functions should be named, one subject responded,

> Mysteriously: 'foo', 'bar.' All those good things. Single letters. So as to not, quote, give away the answer.

Meanwhile, on the good style end, another interviewee cleanly states the tension: "I struggle with this one, right, because, we talk a lot about the names of the variables being useful, right, to the person who's reading the code, and we ask them to do it..."

Multiple subjects indicated that naming could be used as a way to modulate question difficulty. EipE questions are easier if the code uses descriptive names and harder if names are obfuscated. To one interviewee, obfuscating names comes down to instructor choice on difficulty and, maybe, even the difficulty of the code itself:

> Yeah, I think that's an instructor choice, where you can modulate the difficulty... if it's a piece of code that is harder for students to understand, I might give them more clues with variable names. Whereas, if I think that it's reasonable for students to be able to understand it, I might give (pause) single letter (pause) non-informative names.

In addition to the idea of names as a difficulty modulator, two subjects suggested approaches to leverage this tension in a pedagogically productive way. One suggestion was to use the names as a scaffolding tool. In this way, EipE questions should use good names when they are first introduced. As students gain practice with them, useful names should be removed, but with a clear justification and explanation:

> Later on, we might say, "Here's a piece of code, you're doing this", you know, "you're doing the explain questions again, we have intentionally made the function name and the variable names not useful", but we'll tell them we're doing it, right, and we're doing it for (pause) "We want you to read this code and understand the pattern", and we'll explain that we do it that way, but early on, everything makes sense [with naming conventions] because we want to be sort of consistent with the modeling that we've been doing and what we've been telling them that we want them to do.

The second approach to leveraging this tension is to turn that tension into a teachable moment during lecture time. EipE questions should always be presented with obfuscated names and class time activities with EipE should be used to talk about the importance of good names for legible code:

> On the contrary, if you handle the teaching right, it encourages them to use meaningful names because you say, after they've answered the question, ... you can then talk about, "Well, how could we make this code much more easy to understand?" And that would be by putting in some meaningful variable names. So you start with the meaningless variables and then move to a more transparent piece of code by adding the meaning later, and I think that's a very powerful lesson.

6 RESULTS: GRADING EIPE QUESTIONS

In this section, we begin with generalities of grading. We first discuss grading in theory before moving on to general practical concerns, including English as a second language. Next, we discuss the generality of EipE rubrics. At the end of this section, we discuss specific grading issues.

6.1 How should we grade EipE questions in theory?

The fundamental challenge of EipE grading is the imprecision of natural language.

> All natural languages are imprecise, unlike code, ... so people will ... make statements that are not 100% precise.

While an experienced programmer may know how to precisely convey an idea and where precision is needed, novices lack that knowledge and, thus, give imprecise descriptions. This imprecision creates ambiguity that a grader needs to grapple with.

> Is the imprecision because [the students] don't understand the concept and they're saying it imprecisely because they don't understand that there's more that's needed, or are they saying it imprecisely because that's how we speak as humans? We speak very imprecisely.

Our participants were split about how to deal with this ambiguity. Roughly half of them felt that their job in grading was to estimate from a student's answer how well the student understood the code and to grade that understanding.

> It isn't a communication activity as much as it is a personal ability to step away and look at design or purpose instead of syntactic things. So for us, it's more that we're grading their understanding of what is happening and less their ability to explain it.

The other half felt that it was important to grade a student's ability to communicate the behavior of the code, because trying to grade a student's understanding was too difficult or prone to bias.

> I think it has to be communication. You know, ... I don't think you can read very much into an answer before you fall into that hole where you're just making up what you think they know, when in fact they haven't shown– they haven't demonstrated that they really know what they're doing. So I think you have to grade on what they say and not what you think they meant.

> I might know they understand– meaning, I might have a top student and know they understand something, but what they deliver actually doesn't demonstrate that... So it's less about how much I know they can make a free throw, but how much did they actually make or miss a free throw.

One interviewee expressed that the decision between these perspectives might be determined by the size of the code segment used. Larger patterns may allow for more leeway for small precision or ambiguity issues.

> It's sometimes really hard to tell apart the two ... If they explain it wrong, so if it was a bigger one, it could be a little bit less exact in terms of how we express the thing ... and the smaller exercises it kind of forces you to look more in the presentation.

There was much agreement that these two perspectives are closely related. One interviewee's immediate response to having to choose between these perspectives was: "I don't know that those are different. It's very subtle." Another interviewee noted a prerequisite relationship between understanding and communication: "I'd like to believe that they have to understand the code in order to be able to communicate that to somebody else."

In contrast, there was universal agreement that describing "corner case" behavior of code wasn't necessary on summative assessments for CS 1 students. When asked, no participant took points off from answer 1c even though it didn't clearly specify what happens when multiple copies of the minimum value appears in the list (i.e., it returns the lowest index of the two).

> So if I were in a higher level course, I would ask for that preciseness, but not in a CS 1. I'm just more concerned that they understand the- you know, generally, what's the behavior.

A few subjects also remarked that they wouldn't penalize students for errors related to things other than coding knowledge (e.g., non-trivial mathematical concepts). In the context of answer 2b, one participant said that they would give the answer full credit, saying:

> I'm not certain that they understand absolute value... but I'm actually not concerned that they don't understand the code.

6.2 Practical concerns of EipE grading

Several interviewees have expressed the idea that they would be more tolerant to ambiguous answers in exam conditions, as exam anxiety and pressure could lead to poor phrasing of an idea. One subject noted, in contrast, that grading could be stricter on a lower-stakes formative assessment to emphasize particular aspects of the problem.

> I certainly don't like to penalize students when I feel that there's poor expression or even some slip of the pen, so to speak, slip of the tongue, under exam conditions.

Some instructors noted that their assignment of partial credit would likely be influenced by the students' performance in aggregate.

> Frankly, everyone's looking to get some sort of a reasonable distribution and a reasonable pass rate out of their class, so you compromise that in the process of getting a pragmatic grade distribution. So I suppose if my pilot marking had found I was getting almost nobody getting this question right, ..., if I was worried that I was going to have very few students getting it right, and I was worried about the overall grade distribution, I might give this half, but it's an unsatisfactory answer.

Others cited fairness as a motivation for assigning partial credit.

> I would go, "Well, okay, so they understand sort of what a list is, they understand that they're checking for equality, and they understand that they're incrementing something", so it feels like giving it zero would be a little unfair.

Faculty noted, however, that one has to be careful in assigning partial credit lest it might encourage students to brain dump or write intentionally brief, ambiguous answers.

> I have a feeling the student is applying good exam technique, which is "when you don't know the answer, put something down and try to make it seem as plausible as possible." ... It makes me feel at the student who's just trying to get something out of me, when they don't really know the answer.

> I think this is tricky because, like, if they had said a little bit more, they might have shown me that they don't get it.

Another practical concern was training teaching assistants (TAs) to grade EipE questions consistently.

> The training for getting those TAs to coordinate and be aligned with each other or even consistent with themselves is difficult.

6.3 Awareness of English as a Second Language concerns

A widely noted confounding factor was the potential for English as a Second Language (ESL) students to run into unique issues while answering EipE questions. Faculty were hesitant to heavily penalize problems they perceived to originate from ESL issues. In particular, answers 2b, 2h, 3a, and 4b brought up ESL concerns.

> Apart from the exam conditions, there's also the issue of just how articulate the student is, especially if English is the second language. All sort of things crop up there ... *(in relation to the "if it is negative" portion of 2b)*

> At this point, my brain is going, "Is this someone who doesn't have English as their first language?" And am I being unfair to them at this point? *(in relation to the "that variable negative" portion of 2h.)*

> I have a lot of folks who are English as a second language folks, so I'm sensitive to people trying to say what they wanna say without saying it the way I would say it. *(in relation to the difference between "whether" and "if" in 3a.)*

> There is potentially an English language issue here, because we sometimes see people pluralizing things like 'values'. Some of my students will actually mean, like, "the value in y", but they come from a language that doesn't pluralize....so we have to be really careful of that here. *(in relation to the "values in y" portion of 4b.)*

6.4 Designing a rubric for EipE

Many subjects noted the importance of having a rubric in order to grade consistently (especially with multiple graders), but when asked whether a rubric needed to be created on a per-question basis or if there was a single, universal EipE rubric, the consensus was that the answer was somewhere in between.

A number of interviewees noted that it was necessary on a per-question basis to decide what should be included for a complete answer.

> I'd start with the perfect answer... and circle parts of it that I'm looking for. ... For each question, you decide what are the pieces that are important.

Having defined a complete answer, there was significant support for a meta-rubric for grading EipE questions.

> I think ideally, you would have a kind of a universal way of coming at them. "This is the way I grade these problems." Y'know, these are the sorts of issues that I see over and over again. I see that your answer is incomplete, I see that your answer is incorrect, I see that your answer is step-by-step when I ask you to give me the general idea. So I think it's possible to have a general approach to grading [EipE].

> I think that sort of like taking something like SOLO and saying, "Okay, what does this level mean for this question?" I think that there's, like, you know, maybe they've all got the same bones.

For assigning partial credit, a universal rubric may be possible, but executing it may require significant pedagogical content knowledge. One subject noted, "Binary, I think you can use a general rubric. I think it's really harder when you start to go percentages." All of our subjects appeared to use a distance metric from gold standard answers to assign partial credit,

> Here is the perfect answer. Here's my answer. As you're farther and farther away, take points off.

and there were aspects of this distance metric that were universal. For example, incomplete answers were graded less harshly than incorrect answers. Other aspects of this distance metric, however, required knowledge of potential student misconceptions, which may need to be explicitly communicated to less experienced graders, even if there is a universal distance metric.

> And maybe you need to be thinking explicitly about what misconceptions are you looking for, not just, like, generically says some thing wrong (pause) But, like, specifically, is there evidence [of the misconception].

In Section 7, we attempt to characterize this universal distance metric by interpreting how our subjects assigned partial credit.

6.5 Specific grading decisions

6.5.1 Relational vs. Correctness. While all subjects agreed that the best answers were both high-level (relational) and correct, there was a notable dichotomy in which aspect was preferable if students only provided one or the other. Some participants preferred correct multi-structural answers to incorrect relational answers, often citing concerns that students might not understand what was expected of them or that many of their students might not be capable of writing relational answers. One participant that gave no credit for answer 1a ("I think it's wrong.") but gave full credit to answer 1b:

> Ah, yes, the concrete... (long pause) Which is correct. ... I think I would give it 100% because it is correct, but it isn't the kind of answer I really would like to see.

One participant viewed lifting a multi-structural understanding to a relational one as taking "that last step" and that writing a relational answer was "what an A student does versus what a B student does."

In contrast, a number of interviewees graded multi-structural answers such as 1b very strictly, giving them no credit whatsoever. These subjects asserted that multi-structural answers defeated the whole purpose of EipE questions. One participant argued that "most people can do a line-by-line description." One interviewee that gave 25% partial credit to 1a and gave no credit to 1b said about 1b:

> The student did not learn what my main goal was in teaching them ... about how to read code and how to answer these types of questions. So I would give this a zero...

Those that were the most harsh about multi-structural answers were also often the most vocal about ensuring that students were well prepared to do EipE questions, suggesting that being unforgiving about multi-structural answers was fair if students had been trained.

> Early on, we do see a lot of this, right, but it's happening in class and lecture, and we tell them this is– "You get a zero for this because you're not actually doing what we're asking you to do." So when it came to a summative point of view, we would give them that zero.

One subject indicated that it was important to not give partial credit for multi-structural answers, because it might cause students that weren't sure of their relational answer to hedge and submit a multi-structural answer to guarantee some points.

A number of subjects noted that mentioning local variables (e.g., "z" in 4c and 4d) was typically indicative of multi-structural answers.

> I don't like this at all. ... Never use a local variable. ... We call this a data abstraction violation. ... What the heck is "z"?

6.5.2 Precision matters, but colloquialisms are acceptable. Precision often came up with respect to answer 1a, where the use of "smallest index" was consistently penalized for an incorrect assembling of key words. One interviewee noted, "I like the word 'smallest'. I like the word 'index'. Unfortunately, not right next to each other." Word order here matters, as it impacts the way the answer is parsed. Returning "the smallest index" has a clear meaning, one that our subjects had trouble ignoring.

> They're not that far off, right, I think they kind of have a clue what this is doing, but they're not being precise enough...they put their English together wrong...

In contrast, there were other times where imprecise answers were accepted, sometimes grudgingly. In these cases, an experienced programmer would make the correct interpretation in spite of the imprecision; one subject referred to these as "colloquialisms." This occurred most notably with answers like 4a, where saying the function "counts ..." implies that the count is returned. Most participants prefer wording like "returns a count of ...", but gave full credit to 4a with sentiments like "I don't love how it's worded, but that doesn't really matter." Similarly, in reference to 2f's use of "x as a positive number", one interviewee states it is not their "favorite way to put it, but I would probably take it on an exam." Some interviewees even left explicit returns off of their own answers when confirming they understood a given piece of code.

One participant notes that it can be hard for students to understand where precision is required and where one can take shortcuts.

> ... the details matter in coding, and the problem is that most people who don't code don't recognize which details matter and which don't.

6.5.3 Returns can be implicit so long as the return type is clear. As noted in the previous paragraph, our subjects accepted student answers that implicitly specified the return.

> I'm trying to figure out, like, if I really need them to say the word return. I don't think I do because ... that's just the process of how computers and functions are working. It doesn't need to be there necessarily to describe what the code does.

Our participants were unforgiving, however, if implicitly specifying the return suggested that the wrong type of value was being returned. Multiple interviewees took umbrage with 5a because "determines" suggests a Boolean return type.

> I like that they're noting that it's comparing "x" to "y" and determining which one is greater, but their answer suggests that they think it's like returning a Boolean. So they're not really noting that it's returning the larger of the two values.

> So I would give this a zero, because I do think it's important to recognize that beyond determining "if x is greater than y", values are returned differently, you return "x" if it's greater and "y" otherwise.

Answer 3a is interesting in this regard, because there are two possible ways to parse the sentence. First, the return itself might be conditional, but without specifying the return value. Second, it could be returning a Boolean indicating membership of an integer in a list. One interviewee who parsed it as the first said:

> They clearly understand that you're looking for 'y' in the list 'x', but they have failed to capture the notion that you're going to return either true or false. So this one is incorrect.

Interestingly, interviewees universally consider the answer correct when "whether" is substituted for "if" in answer 3a. Whether seems to unambiguously indicate a Boolean return type.

> Oh, that would have been 100%. Yes. 'Cause 'whether' is a true or false? Yeah.

Another interviewee prefers "whether" to "if" also because it abstracts the student answer from the language keywords present in the provided code, suggesting relational reasoning.

> What I like about that answer, "whether," is that it has substituted the keyword "if" ... When I see a student move away from keywords and what have you, I see some sort of abstraction process going on. ... My initial interpretation of that answer [with "if"] is that they're pretty much telling me what that one third line of code does; they're just saying in words that the third line does.

Intriguingly, there was no consensus among our subjects as to the return type of "find out" in answer 5c. In general, 5c earned more partial credit than 5a, indicating that there was more ambiguity of its return type. Some participants scored 5c completely correct.

6.5.4 The computation's inputs can also be implicitly specified. Like return values, our interviewees prefer students to be explicit about the inputs to the computation. Some prefer students to include the variable names of input values directly in their answer. Others prefer that inputs are specified by role (e.g., "a given list"). Most of the time, there is little to no penalty for not explicitly specifying variable names so long as the inputs can be mapped unambiguously to the provided code.

Furthermore, our subjects were generally accepting of implicitly specifying the functions inputs. Answer 5b "returns the largest number" explored this issue because it leaves the domain of the computation completely unspecified. While the student's answer leaves open the possibility that the input could be something like the set of all representable numbers, none of our participants considered that possibility. Almost all participants gave this answer full credit. The two exceptions gave it a large amount of partial credit, penalizing the use of "largest" instead of "larger".

> I would give them ... 80%. I would have preferred if they had said, "returns the larger of x and y." "Largest" implies more than two.

7 DISCUSSION

7.1 What process are graders using to interpret EipE answers?

Looking at the collection of scores that our participants assigned holistically, we can attempt to make inferences about how experienced educators approach EipE grading. We liken it to Bayesian statistical inference, where a grader has a prior probability distribution of possible errors that a student could make that influences their interpretation of the correctness of ambiguous student answers. Mostly notably, our experienced instructors had knowledge of common student misconceptions and English competency that was triggered by some student answers. If we are accustomed to

students speaking in a certain way, we are more likely to assume they know what they are doing even if they leave some of the finer details out. Similarly, if we are attuned to students making certain mistakes in their code, then we are predisposed to notice and punish those errors so that students correct their misunderstandings.

Answer 4b ("count how many values in x are equal to **values** in y") illustrates this phenomena clearly. There are three interpretations of the second "values" (which should be the singular "value") in that answer: (1) it is a typo and inconsequential, (2) it is a grammar or ESL issues and not an understanding issue, or (3) students misconceive 'y' as a list. Most participants initial interpretation is the third one (type error); we presume that this is from lots of prior experience with students failing to properly distinguish between types. While some harshly penalize this as a serious correctness issue, others then consider a second interpretation as an ESL issue (as noted in Section 6.3). Those that arrive at this second interpretation generally are more accepting of this answer, scoring it correctly or nearly correctly. While the misspelling/typo in answer 4c ("everytime") received no comment or penalty, every one of our participants commented on "values", but none of them indicated that "values" could have been a typo.

7.2 Is a universal distance metric for partial credit possible?

Overall, we saw trends in how our subjects assigned partial credit that suggest that our community might be able to agree on a universal "how far is this student answer from my gold standard answer?". We discuss the general trends that lead to this conclusion.

First, on the issue of scoring multi-structural answers raised in Section 6.5.1, we believe that if students are given sufficient instruction on and practice with EipE questions before assessment and multi-structural answers are graded consistently throughout the semester, then it is likely acceptable to assign them low scores. Furthermore, we agree with one subject that a student answer that includes both a multi-structural and relational answer together can be graded as a relational answer.

Second, we observed that incomplete or ambiguous descriptions of code generally received more partial credit than perceived incorrect answers. Ambiguous description leaves room for graders to assume a student knows what is going on or generously assume that unstated details are implicit, rather than the alternative explanation that students may not actually understand the code. Scores for incomplete answers were generally proportional to completeness, with answers that were closer to fully specified receiving more partial credit than answers that left out more significant pieces.

Third, we found that subjects penalized answers that suggested that the student didn't understand the language more harshly than answers that suggested that the student didn't understand the code. The clearest example of this is the suggestion that the function returns multiple times in answer 2f, which was typically assigned very low scores.

> What's really wrong (pause) is then they go on to say, "then return x as a negative," and you only return once. So they clearly do not understand the nature of returns.

Another instance of a perceived language misconception being penalized harshly came from answer 2c's "make x positive", where some subjects perceive that the student believes that the contents of variable x is being updated rather than a new value being returned.

> No, I think I would fail that actually, 'cause I think that shows a fundamental misunderstanding about what functions do.

In contrast, answers that were strictly incorrect, but where the issue was perceived to lie with understanding that specific code and not the language, were more likely to receive partial credit. Many subjects engaged in a practice that we referred to as *some good, some bad* to assign non-trivial amounts of partial credit. For example, even though 1a's "the smallest index" was routinely considered incorrect on a binary scale, the presence of "smallest" and "index" led many to give the answer around half credit.

Establishing this universal distance metric would support the construction of NLP autograders. Given consistent and scalable scoring presents a barrier to the use of EipE questions, an NLP autograder built upon an agreed universal standard could be widely adopted as a way to utilize EipE in classes. Even if such an autograder's scores are not assigned exactly as an instructor might have chosen, machine scoring removes the inconsistency from having multiple graders.

8 LIMITATIONS

We identified three main limitations with our work: subject pool, answer selection, and answer ordering. First, due to the labor intensity of the qualitative research methods used, we only interviewed 11 participants. While we have some confidence that we sampled to saturation, we cannot be sure that with additional subjects or different subjects that our conclusions might not be somewhat different.

Second, in hindsight, our selection of student answers incompletely characterized the partial orderings for the distance metric discussed in Section 7.2. For example, none of the answers we presented were relational but egregiously wrong. As such, we are unable to compare the community's opinions on egregiously wrong relational answers versus perfectly correct multi-structural answers, which would shed further light on the correctness vs. relational issue.

Additionally, our collection of answers included only one example of an innocuous spelling or grammar mistake (4c's "everytime"). While we believe that our interviewees would consider them as inconsequential, a single example provides little support for that assertion, and we don't know if there is a point at which having too many such mistakes has consequences.

Lastly, our study is potentially vulnerable to ordering effects. While not every interviewee saw every answer, either due to selection on our end or running out of time during the interview, we did present questions in mostly the same order across the interviews. If there is an ordering effect that results from seeing one answer before another — for example, 5a before 5b and 5c — our results do not allow us to account for that. It is unclear to us that would have led to any major differences in observed themes, but future studies on EipE grading should control for that possibility.

9 CONCLUSION

In this paper, we attempted to tap into the introductory programming research community's expertise around "Explain in plain English" code reading questions. We believe that we've synthesized this knowledge so as to identify a series of suggested practices relating to their use and grading in CS 1 courses. In particular, we provide guidance on the selection and presentation of EipE questions, on training of students to complete EipE questions, and on the integration of EipE questions into courses. To follow best practice, EipE questions should be summarizable segments of code, not so small that there is no difference between relational and multi-structural answers but not so large as to defy clear and concise description. Introductory patterns may be some of the best code to use for EipE questions given their size and explainability. Instructional time must be spent training students how to answer EipE questions and grading expectations must be clearly laid out to students. If utilized well, EipE may both serve as excellent formative activities for students as well as a productive way to getting students thinking about abstraction in programming by having them recognize code snippets with clear, summarizable purposes.

In addition, we discuss grading of EipE questions on both theoretical and practical levels and characterize the community's standards relating to the relative value of different kinds of answers. Relational answers are preferable to multi-structural answers. Incompleteness and ambiguity tend to be more acceptable than incorrectness. Small errors and possible typos are often permitted, but apparent misconceptions about the programming language are harshly punished. From these standards, it may be possible to define a universal distance metric from students' answers and an instructor's gold answer. In turn, establishing these standards and this metric may allow for NLP autograders to be developed and used to empower larger classes to use EipE questions in a scalable fashion.

Our interviews paint a picture of EipE questions being underutilized by the community. While our interviewees overwhelmingly perceived value in EipE questions, they rarely used EipE questions in their own teaching to a commensurate degree. We hope that this distillation of knowledge is useful both for instructors who want to introduce EipE questions into their courses for the first time as well as tool/content builders that will build EipE resources for use by others.

ACKNOWLEDGMENTS

Our deepest gratitude goes to the members of the community that shared their time and insights with us.

REFERENCES

[1] Owen Astrachan and David Reed. 1995. AAA and CS 1: The Applied Apprenticeship Approach to CS 1. In *Proceedings of the Twenty-sixth SIGCSE Technical Symposium on Computer Science Education* (Nashville, Tennessee, USA) *(SIGCSE '95)*. ACM, New York, NY, USA, 1–5. https://doi.org/10.1145/199688.199694

[2] Sushmita Azad, Binglin Chen, Maxwell Fowler, Matthew West, and Craig Zilles. 2020. Strategies for Deploying Unreliable AI Graders in High-Transparency High-Stakes Exams. In *International Conference on Artificial Intelligence in Education*. Springer, 16–28.

[3] Walter Beck, S. Rebecca Thomas, Janet Drake, J. Philip East, and Eugene Wallingford. 1996. Pattern Based Programming Instruction. In *1996 Annual Conference*. ASEE Conferences, Washington, District of Columbia. https://peer.asee.org/6228.

[4] John B. Biggs and K. F. Collis. 1982. *Evaluating the quality of learning : the SOLO taxonomy (structure of the observed learning outcome) / John B. Biggs, Kevin F. Collis.* Academic Press New York. xiii, 245 p. : pages.

[5] Binglin Chen, Sushmita Azad, Rajarshi Haldar, Matthew West, and Craig Zilles. 2020. A Validated Scoring Rubric for Explain-in-Plain-English Questions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE)*.

[6] Michael J. Clancy and Marcia C. Linn. 1999. Patterns and Pedagogy. In *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education* (New Orleans, Louisiana, USA) *(SIGCSE '99)*. ACM, New York, NY, USA, 37–42.

[7] Malcolm Corney, Sue Fitzgerald, Brian Hanks, Raymond Lister, Renee McCauley, and Laurie Murphy. 2014. 'Explain in Plain English' Questions Revisited: Data Structures Problems. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. ACM, New York, NY, USA, 591–596. http://doi.acm.org/10.1145/2538862.2538911

[8] Malcolm Corney, Raymond Lister, and Donna Teague. 2011. Early Relational Reasoning and the Novice Programmer: Swapping As the "Hello World" of Relational Reasoning. In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114* (Perth, Australia) *(ACE '11)*. 95–104.

[9] Adrienne Decker, Lauren Margulieux, and Briana Morrison. 2019. Using the SOLO Taxonomy to Understand Subgoal Labels Effect in CS1. *ICER'19 - Proceedings of the 2019 ACM Conference on International Computing Education Research*, 209–217. https://doi.org/10.1145/3291279.3339405

[10] Max Fowler, Binglin Chen, Sushmita Azad, Matthew West, and Craig Zilles. 2021. Autograding "Explain in Plain English" questions using NLP. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 1163–1169. https://doi.org/10.1145/3408877.3432539

[11] Fernand Gobet, Peter CR Lane, Steve Croker, Peter CH Cheng, Gary Jones, Iain Oliver, and Julian M Pine. 2001. Chunking mechanisms in human learning. *Trends in cognitive sciences* 5, 6 (2001), 236–243.

[12] Vighnesh Iyer and Craig Zilles. 2021. Pattern Census: A Characterization of Pattern Usage in Early Programming Courses. In *Proceedings of the SIGCSE Technical Symposium (SIGCSE)*.

[13] Joe Bergin. 1998. Patterns for Selection. http://csis.pace.edu/~bergin/patterns/Patternsv5.html.

[14] Raymond Lister. 2020. *On the Cognitive Development of the Novice Programmer: And the Development of a Computing Education Researcher.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3442481.3442498

[15] Raymond Lister, Elizabeth S Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. 2004. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin* 36, 4 (2004), 119–150.

[16] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further Evidence of a Relationship Between Explaining, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education* (Paris, France) *(ITiCSE '09)*. ACM, New York, NY, USA, 161–165. https://doi.org/10.1145/1562877.1562930

[17] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further Evidence of a Relationship Between Explaining, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education* (Paris, France) *(ITiCSE '09)*. ACM, New York, NY, USA, 161–165. https://doi.org/10.1145/1562877.1562930

[18] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the Fourth International Workshop on Computing Education Research*. ACM, 101–112.

[19] Sandra P Marshall. 1995. *Schemas in problem solving*. Cambridge University Press.

[20] Katherine B McKeithen, Judith Spencer Reitman, Henry H Rueter, and Stephen C Hirtle. 1981. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology* 13, 3 (1981), 307–325.

[21] Laurie Murphy, Sue Fitzgerald, Raymond Lister, and Renée Mccauley. 2012. Ability to 'Explain in Plain English' Linked to Proficiency in Computer-based Programming. *ICER'12 - Proceedings of the 9th Annual International Conference on International Computing Education Research.* https://doi.org/10.1145/2361276.2361299

[22] Laurie Murphy, Renée McCauley, and Sue Fitzgerald. 2012. 'Explain in Plain English' Questions: Implications for Teaching. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. ACM, New York, NY, USA, 385–390. https://doi.org/10.1145/2157136.2157249

[23] Owen Astrachan and Eugene Wallingford. 1998. Loop Patterns. https://users.cs.duke.edu/~ola/patterns/plopd/loops.html.

[24] Robert S Rist. 1989. Schema creation in programming. *Cognitive Science* 13, 3 (1989), 389–414.

[25] John Sweller. 2011. Cognitive Load Theory. In *Psychology of learning and motivation*. Vol. 55. Elsevier, 37–76.

[26] John Sweller, Jeroen JG Van Merrienboer, and Fred GWC Paas. 1998. Cognitive architecture and instructional design. *Educational psychology review* 10, 3 (1998), 251–296.

[27] Anne Venables, Grace Tan, and Raymond Lister. 2009. A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer. In *Proceedings of the Fifth International workshop on Computing Education Research*. ACM, 117–128.

[28] Eugene Wallingford. 1996. Toward a First Course Based on Object-oriented Patterns. In *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education* (Philadelphia, Pennsylvania, USA) *(SIGCSE '96)*. ACM, New York, NY, USA, 27–31. https://doi.org/10.1145/236452.236485

[29] Jacqueline Whalley, Raymond Lister, Errol Thompson, Tony Clear, Phil Robbins, P K Ajith Kumar, and Christine Prasad. 2006. An Australasian study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. *Eighth Australasian Computing Education Conference (ACE2006)* (2006).

[30] Susan Wiedenbeck. 1985. Novice/expert differences in programming skills. *International Journal of Man-Machine Studies* 23, 4 (1985), 383 – 390. https://doi.org/10.1016/S0020-7373(85)80041-9

[31] Leon E. Winslow. 1996. Programming Pedagogy— a Psychological Overview. *SIGCSE Bull.* 28, 3 (Sept. 1996), 17–22. https://doi.org/10.1145/234867.234872