ReDefining Code Comprehension: Function Naming as a Mechanism for Evaluating Code Comprehension

David H. Smith IV
University of Illinois
Urbana, USA
dhsmith2@illinois.edu

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

Paul Denny
University of Auckland
New Zealand
p.denny@auckland.ac.nz

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

Abstract

"Explain in Plain English" (EiPE) questions are widely used to assess code comprehension skills but are challenging to grade automatically. Recent approaches like Code Generation Based Grading (CGBG) leverage large language models (LLMs) to generate code from student explanations and validate its equivalence to the original code using unit tests. However, this approach does not differentiate between high-level, purpose-focused responses and low-level, implementation-focused ones, limiting its effectiveness in assessing comprehension level. We propose a modified approach where students generate function names, emphasizing the function's purpose over implementation details. We evaluate this method in an introductory programming course and analyze it using Item Response Theory (IRT) to assess the difficulty and discrimination of function naming exercises as exam items and to compare their alignment with traditional EiPE grading standards. We also publish this work as an open source Python package for auto-grading EiPE questions, providing a scalable solution for adoption.

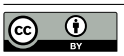
CCS Concepts

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

Keywords

GPT-4o, LLM, Large Language Model, Code Comprehension, SOLO Taxonomy, Explain in Plain English, EiPE, Function Naming, Explain in Plain Language, EiPL

ACM Reference Format:
David H. Smith IV, Max Fowler, Paul Denny, and Craig Zilles. 2025. ReDefining Code Comprehension: Function Naming as a Mechanism for Evaluating Code Comprehension. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2025), June 27-July 2, 2025, Nijmegen, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3724363.3729097



ITiCSE 2025, Nijmegen, Netherlands

ACM ISBN 979-8-4007-1567-9/2025/06
https://doi.org/10.1145/3724363.3729097

1 Introduction

As we continue to transition from the age of traditional programming to the new age of Human-GenAI collaborative coding, it is essential that we prepare students with the skills necessary to succeed in such collaborations. In this transition from traditional CS1 to CS1-LLM [25], instructors have begun to discuss a possible shift towards emphasizing the skills of describing computational tasks through natural language [6, 20] and comprehending code produced by an LLM to verify its correctness [17, 18]. One existing approach for assessing these skills is the use of "Explain in Plain English" (EiPE) questions, in which students describe the purpose of a given code snippet [9]. However, their adoption has been limited by the time-intensive grading process of marking short answer responses.

Though prior autograding approaches developed for EiPE questions have seen some success, they each suffer from distinct limitations. A logistic classifier developed by Fowler et al. [8] matches the performance of a trained human grader but requires a large corpus of human labeled data for the creation of each question. To overcome these drawbacks, Smith and Zilles [23] proposed Code Generation Based Grading (CGBG), an autograding approach that uses a large language model (LLM) to generate code from a student's description and verifies its functional equivalence to the code the student was describing with unit tests. However, this approach struggles to distinguish between high-level, purpose-focused, and low-level, implementation-focused code descriptions.

To address the limitations of prior approaches, we propose a modified approach to CGBG EiPE questions where, rather than having students provide an unbounded description of code, we ask that they provide a syntactically valid Python function name. We hypothesize that the more bounded nature of the entry will nudge students towards high-level descriptions. To evaluate this approach we explore the following research questions:

RQ1: What are the psychometric properties of function redefinition EiPE questions when used in an exam setting?

RQ2: What is the alignment between the autograding mechanism for function redefinition EiPE questions and the grading standards of standard EiPE questions?

Additionally, to allow for the easy adoption of this grading approach, we release it as a new feature in the `eiplgrader`[1] package, an open source Python package for autograding EiPE questions.

[1] https://github.com/CoffeePoweredComputers/eiplgrader

2 Background

To contextualize this work we first provide an overview of Explain in Plain English (EiPE) questions in Section 2.1, discussing their importance and approaches to grading them. In Section 2.2, we discuss autograding approaches relevant to EiPE questions.

2.1 Traditional EiPE Questions

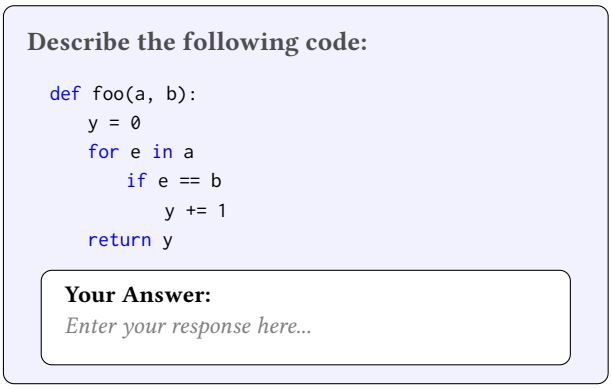


Figure 1: Sample interface for an Explain in Plain English (EiPE) question.

Explain in Plain English (EiPE) questions (Figure 1) are a common approach for assessing and developing code comprehension skills. In these questions, students are shown a segment of code and asked to provide a high-level, natural language description of what that code is doing.

With respect to what differentiates ideal from suboptimal EiPE responses, the "Structure of the Observed Learning Outcome" (SOLO) taxonomy [2] has often been used to categorize responses based on their demonstrated level of comprehension. An adapted version of this taxonomy, introduced by Lister et al. [15], defined the following levels of comprehension for EiPE responses:

- **Relational:** The student's response demonstrates an understanding of the relationships between the code's components by describing the code's purpose rather than its implementation.
- **Multistructural:** The student's response demonstrates an understanding of the code's components but not the relationships between them. This often manifests as line-by-line descriptions of the code.
- **Unistructural:** The student's response demonstrates an understanding of some components of the code but is either missing components or contains errors.
- **Prestructural:** The student's response does not contain any relevant or correct descriptions of the code.

Clear et al. [5] further refined this taxonomy by introducing the categories of *Relational Error* and *Multistructural Error* to account for responses that demonstrate a high-level understanding of the code's purpose or provide a full description of the code, respectively, but contain errors.

Prior work on the topic of EiPE—and code comprehension more generally—has indicated a correlation between students who demonstrate high-level code comprehension skills (i.e., are able to "see the forest for the trees" [15]) and students who are able to succeed on code writing tasks [10, 14, 16]. However, eliciting high-level responses may not always be straightforward. For example, Sheard et al. hypothesized that some multistructural responses may stem from a misunderstanding of the instruction to "explain in plain English", or to "explain the purpose" of code [21]. To address this, they suggested that asking students to nominate a function name might be a promising alternative, as a way of guiding them towards relational thinking. However, to our knowledge, this approach has not been studied in a research setting. Additionally, prior studies have highlighted that expressing ideas clearly in English can be challenging for some students, particularly in linguistically diverse contexts, where language issues may affect their ability to articulate high-level responses [13].

In proposing their theory of instruction for introductory programming skills, Xie et al. [27] suggest that code comprehension is a critical skill, and one that should be taught immediately prior to code writing to improve success in those tasks. In the context of GenAI assisted programming, many have suggested that the ability to articulate computational tasks through natural language, as well as to comprehend code, will both play more critical roles in computing education given the alignment between these skills and the skills of prompting and verifying code produced by LLMs [6, 11, 17, 18, 20].

2.2 Auto-grading EiPE Questions

Despite the clear importance and utility of EiPE questions, their adoption has been limited by the time-consuming nature of grading free-response questions [9]. To address this challenge, several autograding approaches have been developed with some success.

Fowler et al. [8] developed a logistic classifier trained on a large corpus of human-labeled responses for each question. This approach has been shown to provide grades that reliably match those of a trained teaching assistant. However, it is limited by two factors. First, there is a significant question authoring overhead, as a large corpus of human-labeled responses must be created *for each question*. Second, though the core goal of providing automated feedback was achieved, the feedback was limited to marking answers as correct or incorrect while providing some sample correct answers.

To address these limitations, Smith and Zilles [23] proposed an autograding approach—Code Generation Based Grading (CGBG)—that leverages a large language model to generate code from a student's description and uses a suite of unit tests to verify the code generated is functionally equivalent to the code the student was describing. Several evaluations of this approach have demonstrated its effectiveness as well as its ability to provide students with feedback through the results of unit-tests and the generated code [7, 12, 22]. Additionally, given that modern LLMs are trained on a wide variety of written languages, their translation capabilities can allow students to provide prompts in languages other than English [19, 22, 24]. However, a limitation of the CGBG approach is that it is unable to distinguish between low-level responses and high-level ones, limiting its effectiveness at evaluating comprehension level.

Table 1: Question IDs and their corresponding code which students were asked to describe

Question	Count Odd Nums in List	Get Numbers Below Threshold	Absolute Values of List	Count Strings of Given Length

Code	def foo(x: List[int]):
	k = 0
	for e in x:
	if e % 2 != 0:
	k += 1
	return k
		def foo(x: List[int], t: int):
		result = []
		for e in x:
		if e < t:
		result.append(e)
		return result
			def foo(x: List[int]):
			for i in range(len(x)):
			if x[i] < 0:
			x[i] = −x[i]
				def foo(x: List[str], n: int):
				k = 0
				for s in x:
				if len(s) == n:
				k += 1
				return k

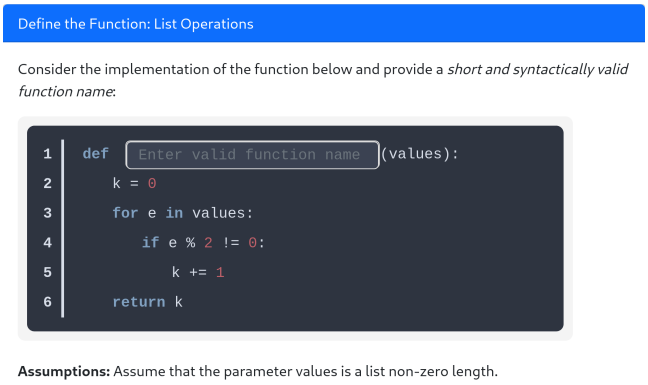


Figure 2: The interface of the function redefinition EiPE task.

3 Methods

To collect student responses for the evaluation of Function Redefinition EiPE questions (Figure 2), we constructed four questions. Students (N=366) from a large, introductory programming course in the United States were each randomly assigned one of these questions on a final exam (Table 1). The exams took place in a proctored computer lab [28] and were administered on the PrairieLearn platform [26].

Responses to these questions had two restrictions: the student's response must be a syntactically valid Python function name and must be no longer than 10 words. In the event a student's submission did not meet these standards, they were informed via an error message and given the opportunity to resubmit without penalty. Students were given three attempts to get the questions correct, each time receiving feedback on the correctness of their response through the standard CGBG approach. In total, we collected 647 responses of which 476 were unique.

To investigate the alignment between the Function Redefinition EiPE questions and the objectives of EiPE questions, we perform an analysis of all student responses collected during the exam. In doing so, we re-grade each response using two approaches:

- **One Attempt Grading:** Here, the students response is used to generate a single Python function. This function is then executed on a set of test cases to determine if the function is correct.
- **Robustness Grading:** This approach uses the student's response to prompt GPT-4o to generate five different implementations of a Python function based on the student's response. The response is considered correct if *all* of the generated functions passes *all* test cases it is run against.

Responses were generated using OpenAI's GPT-4o model with a temperature of 0 to ensure deterministic responses.

We compare the outcomes with respect to the psychometric properties of the questions as they were graded under each of these approaches (Section 4.1) and the analysis of responses using the SOLO taxonomy (Section 4.2) which are discussed in the following subsections.

4 Results

We first look at descriptive statistics for each question—namely response length and correctness—to gain a high-level understanding of student responses to each of the questions.

As expected, due to the 10-word limit imposed by the function redefinition exercises, no responses exceeded this limit. The vast majority of the responses were far shorter, with none of the questions exceeding four words as their average length (Figure 3).

As for performance, there was significant variance in the apparent difficulty of each question (Figure 4). *Count Strings of Given Length* and *Count Odd Nums in List* students submitted more correct rather than incorrect responses. In contrast, *Get Numbers Below Threshold* was more difficulty with the proportion of correct responses only slightly exceeding the proportion of incorrect responses. Finally, *Absolute Values of List* was by far the most difficult with the proportion of incorrect responses far exceeding those that were graded as correct.

4.1 RQ1) Item Response Theory

For our analysis, we employ a two parameter logistic (2PL) item response (IRT) model [3].

$$P(\theta) = \frac{1}{1 + e^{a_i(\theta - b_i)}}$$

We use this model to predict the probability of a student of a given ability level θ responding correctly to an item. The item statistics are as follows:

- a_i The discrimination of item i. Discrimination is the slope of the logistic curve at the intercept with a 50% probability of a correct response and characterizes the ability of the item to discriminate between students above and below that ability level.
- b_i The difficulty of item i. Difficulty is characterized as the ability level (θ) at which a student has a 50% chance of responding to a question correctly.

A shortcoming of traditional 2PL is that it assumes all questions were dictomously graded—an assumption that does not hold with the data collected in this study given that students were allowed

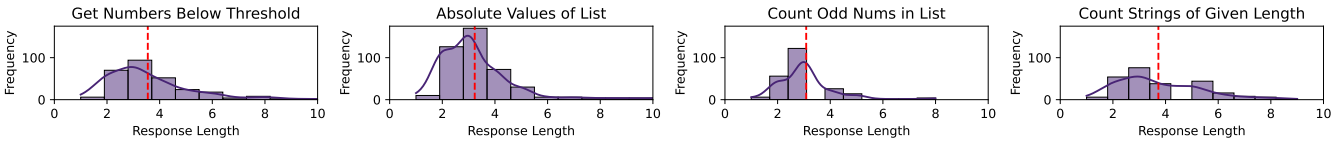


Figure 3: Lengths of all responses (N=647) submitted for each question

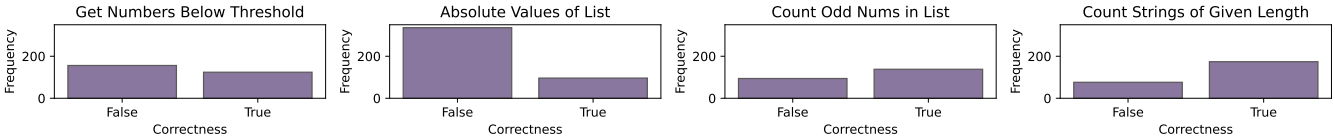
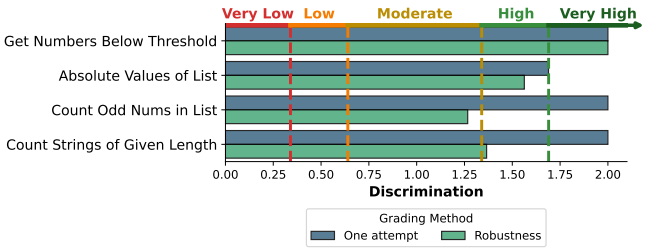
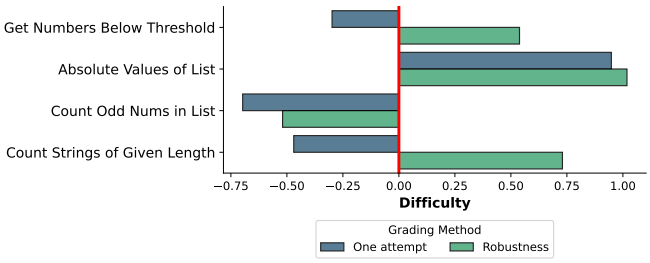


Figure 4: Correctness of all responses (N=647) for each question.



(a) Discrimination (a) Coefficients



(b) Difficulty (b) Coefficients

Figure 5: Item Statistics from the Results of Fitting 2PL IRT

multiple attempts and awarded partial credit through a variety of mechanisms (e.g., partially passing unit tests). To avoid the potential loss of information that we may incur through an alternative method of dicotomizing—such as rounding or only considering the first attempt—we use an adapted version of the cross entropy optimization approach introduced by Chen et al. [4]. In keeping with their approach we also adopt the standard bounds of each of the model's coefficients: $\theta \in [-3, 3]$, $a \in [0, 2]$, $b \in [-3, 3]$. We discuss the results of this analysis in the following subsection. Additionally, to aid in the interpretation of the discrimination coefficients we use the cutoffs presented by Baker [1] of: Very Low (≤ 0.34), Low ($0.35 - 0.64$), Moderate ($0.65 - 1.34$), High ($1.35 - 1.69$), and Very High (> 1.7).

4.1.1 Results. Overall, the results of the IRT analysis indicate that the Function Redefinition EiPE questions function well as exam items, exhibiting high discrimination (Figure 5a) with all questions exceeding the *moderate* threshold and many exceeding *high* and *very high*. In terms of differences in these outcomes between the two grading approaches, we observe that, in three cases, questions graded under the ***One Attempt*** approach exhibited a higher discrimination coefficient than those graded under the ***Robustness*** approach. In two cases, this difference was substantial, lowering the coefficient from well exceeding the *very high* threshold to the upper end of *moderate* and the lower end of *high*.

The primary differences between the ***One Attempt*** and ***Robustness*** grading approaches appears with respect to the difficulty (Figure 5b). In the ***One Attempt*** approach, the difficulty of the questions is relatively low, with only one question—*Absolute Values of List*—achieving a positive difficulty coefficient. Conversly, under the ***Robustness*** approach, all questions except one—*Count Odd Nums in List*—exhibit a positive difficulty coefficient.

> **Key Takeaways:** Both approaches produce items that effectively discriminate between students of different ability levels. The primary consideration for instructors is the trade-off between difficulty and discrimination. The ***One Attempt*** approach produces items that are easier for students, while the ***Robustness*** approach produces items that are more difficult with a small reduction in discrimination.

4.2 RQ2) Alignment with Traditional EiPE

To analyze the quality of the responses to the Function Redefinition EiPE questions—and by extension their alignment with the objectives of EiPE questions—we conducted a qualitative analysis of the responses using the modified SOLO taxonomy presented by Clear et al. [5]. Two researchers independently coded the responses into the categories of *Relational*, *Relational Error*, *Multistructural*, and *Multistructural Error*, or *Other Error*. Agreement was calculated using Cohen's κ, which was found to be acceptable at a value of 0.75, and any disagreements were resolved through discussion.

4.2.1 Results. Overall, the results of the comparison between the Function Redefinition EiPE autograding approach and the SOLO taxonomy indicate that the function redefinition method often aligns

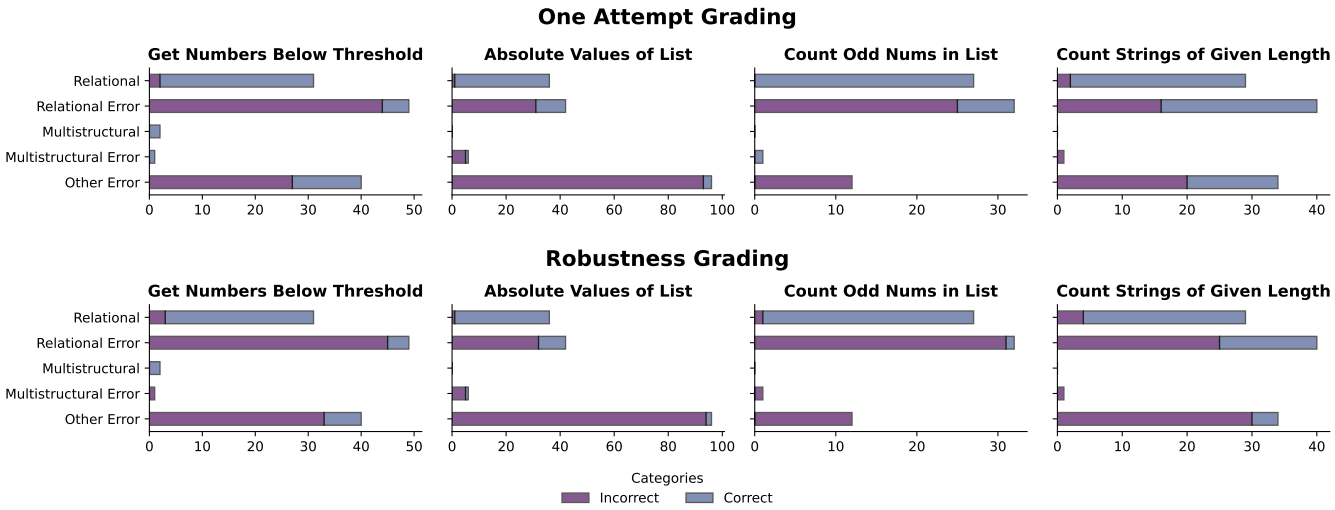


Figure 6: The percentage of responses belonging to each SOLO taxonomy category graded as correct for each grading approach.

with the objectives of EiPE questions. Across all questions—and in both grading approaches—the vast majority of responses categorized as *Relational* were graded as correct. Additionally, given the near complete absence of *Multistructural* responses it would appear that the question format was successful in dissuading students from providing such responses.

Comparing the two grading approaches, in all cases, the use of the ***Robustness*** grading approach resulted in a higher proportion of *Relational Error* and *Other Error* responses being graded as incorrect, thus reducing the prevalence of false positives, while having a very minimal impact in terms of increasing the prevalence of false negatives (i.e., *Relational* responses graded as incorrect).

To characterize the alignment and misalignments that exist between this activity and its grading approaches, we wish to highlight two categories: 1) relational responses graded as correct and 2) relational errors and other errors graded as correct. In doing so, we aim to characterize the nature of what we regard as true positives and false positives due to their misalignment with the SOLO taxonomy.

Relational Responses Graded as Correct: *Relational* responses graded as correct included `get_numbers_under_threshold` and `count_odd_nums`. Such responses are relational and precise, as they describe the action being performed by the function (e.g., getting, counting), the property of the object being acted upon (i.e., values, nums), and, in the case of the former, the condition under which the action is performed (i.e., below a threshold). Such responses were consistently graded as correct under both the ***One Attempt*** and ***Robustness*** grading approaches.

Relational Errors and Other Errors Graded as Correct: In contrast to *Relational* responses, *Relational Error* responses which were graded as correct often missed details. For example, with regard to the *Count Strings of Given Length* question, the response `count_strings` is correct in that it describes the action being performed but fails to provide the necessary context of what is being counted. Similarly a response from the *Other Error* category which was graded as correct, `count_same_length` is correct in that it describes the action being performed, but it fails to provide the necessary context of what is being counted and what is meant by `same`. These absences make it difficult to determine the purpose of the function without additional context and, as such, do not offer a clear indication that the student providing the response has a complete understanding of the code that they are describing. The use of the ***Robustness*** grading approach did successfully filter out the first example but not the second. These results suggest that the combination of the student-provided function name along with the question author provided parameters and assumptions was sufficient for the LLM to accurately "guess" the correct code.

Such good guesswork on the part of the LLM was not limited to this question and appeared to consistently be the reason for questions in *Other Error* and *Relational Error* categories being graded as correct. Other examples include: `less_than_t` for *Get Numbers Below Threshold* and `find_distance_to_zero` for *Absolute Values of List*.

> **Key Takeaway:** Overall, the Function Redefinition EiPE questions exhibit a high degree of alignment with the objectives of EiPE questions and appear to have deterred the use of multistructural responses. Of the two grading approaches evaluated, the ***Robustness*** approach proved an essential—though admittedly not foolproof—mechanism for mitigating the risk of false positives while having a minimal impact on true positives.

5 Discussion

Function naming exercises present an effective and exciting way to assess students' ability to comprehend code. Given the constrained response, compared to traditional EiPE, the question type almost totally avoids multistructural responses, keeping students' answer

efforts more relational. In terms of summative assessment, the exercises have at least moderate, and generally high, discrimination, providing value as test items. While there is a trade-off in difficulty behind whether one chooses to use one-attempt or robustness grading, using the latter reduces the risk that GPT-4o, capable as it is, constructs successful functions from answers with small errors. This reduces the number of relational errors marked correct by the grader, with only *Count Strings of Given Length* showing less success. We attribute this difference in performance to the structure of the question. Considering an example relational error `count_strings`, our grading paradigm and the student's answer provides the action (counting), data types (strings), function definition (count_strings), and question assumptions (the question takes a list and an integer). This may be a more trivial problem for an LLM to fill in the blanks to generate successful functions more often, if only because the integer parameter implies some actions with that integer, for which a comparison with string lengths is one of the few reasonable assumptions the model could make.

In light of function naming exercises constraining students more towards relational responses, we are particularly excited by the potential for these exercises to serve as scaffolding for building students' competence for traditional EiPE exercises and, broadly, the skills of comprehending and explaining pieces of code. It may be that having to describe a piece of code in a few key words is a useful precursor to being able to reason about and describe the code at a high-level of abstraction. Summatively, this could provide a more gradual progression of students' code comprehension ability: rather than succeeding or failing at an EiPE task, it may provide an intermediate skill of appropriate function naming. Formatively, it may be that function naming is a more accessible task earlier in the semester of a programming course and, as students develop their programming and code comprehension skills, function naming tasks can lead into more traditional, high-level EiPE tasks. One avenue future work could explore is using the same functions for function naming exercises and traditional EiPE exercises to directly compare the difficulty between the two and further investigate this potential relationship between the skills.

The robustness grading method for these questions has additional potential benefits beyond reducing false positives in the grading process. Robustness grading may itself prove to be a valuable partial credit mechanism, which is an open question that plagues the use of EiPE exercises [9]. Students can receive points not just for generating a correct function, but for how many correct functions their proposed function name generates. This can still be automated with the same test cases, but allows for more granular scoring and feedback to students. Function names that work five out of five times are likely higher quality than function names that happen to work once out of five times, which itself is more feedback than all-or-nothing correctness grading. This may also have the benefit of making students happier, as some points are better than no points.

Finally, we wish to briefly comment on the large number of relational errors that we noted in students' responses, especially in light of the false positive count for *Count Strings of Given Length*. In applying the SOLO taxonomy, our standard was arguably maximally harsh in that we leveraged our existing knowledge of applying the taxonomy to *full EiPE responses* and only considered the function names, not the other assumptions and information available. While EiPE responses are ideally short, the size of a function is shorter still. The function name `count_strings` may imply something along the lines of "a function that counts all the strings in a mixed-type list," in a vacuum. However, in the context of knowing the function takes a list of strings and an integer, it may be reasonable to take the name *and* the arguments as sufficient to mark the answer as relational *without* an error. Other researchers may have ended up with a lower count of relational errors with a less strict yet still reasonable standard. There may also be value in adapting the SOLO taxonomy for use with function name sized responses, especially in light of multistructural responses barely existing.

6 Limitations

There are some key limitations that we wish to note. First, while we consider *Count Strings of Given Length* to be an outlier that performs more poorly than our other questions, it is also possible that the other three questions may instead be outliers that perform unusually well for this task. We cannot be certain our findings generalize to all code that could be used for function naming exercises. Future work that investigates more code snippets will be useful for refining our understanding of the exercises' difficulty and discrimination, i.e. their value as test items.

There are two course specific limitations we wish to raise. First, students in the class saw these exercises relatively late into the semester: for the first time on homework during week 12 of a 16 week course. This means that students did not have the whole semester to familiarize themselves with these questions and their performance will likely differ when given longer periods of time to work with this style of question. On the other hand, students in this course are exposed to traditional EiPE exercises regularly from the first week of the class onward, with exercises focused on small functions being commonplace from the fourth week onward. Given this, the students in this course may have been more suited to function naming exercises than typical CS0/CS1 students. In turn, studies with different student populations may find higher difficulty measures than we found here if their student population is less familiar with code comprehension being assessed.

7 Conclusion

We propose and evaluate function naming exercises as an approach to assessing code comprehension skills. Ultimately, these exercises show promise as test items, featuring mostly high discrimination as well as the ability to scale difficulty through the selection of the easier one-attempt or harder robustness (e.g. generate five times, not once) grading methods. Function naming exercises present high quality, easy to construct exam items. Further, they have exciting potential as a scaffolding step in the development of code comprehension and are worthy of future study.

While future work should evaluate more code snippets for use as function naming exercises to determine whether our findings generalize further, initial signs for this type of exercise are promising. Readers interested in deploying their own function naming exercises can use our feature in the `eiplgrader` package to ease adoption and deployment.

References

[1] Frank B Baker. 2001. *The basics of item response theory*. ERIC.

[2] John B Biggs and Kevin F Collis. 2014. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press.

[3] Allan Birnbaum. 1968. Some latent trait models and their use in inferring an examinee's ability. *Statistical theories of mental test scores* (1968).

[4] Binglin Chen, Colleen M Lewis, Matthew West, and Craig Zilles. 2024. Plagiarism in the age of generative ai: cheating method change and learning loss in an intro to CS course. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 75–85.

[5] Tony Clear, Jacqueline Whalley, RF Lister, Angela Carbone, Minjie Hu, Judy Sheard, Beth Simon, and Errol Thompson. 2008. Reliably classifying novice programmer exam responses using the SOLO taxonomy. *National Advisory Committee on Computing Qualifications* (2008).

[6] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A Becker, and Brent N Reeves. 2024. Prompt Problems: A new programming exercise for the generative AI era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 296–302.

[7] Paul Denny, David H Smith, Max Fowler, James Prather, Brett A Becker, and Juho Leinonen. 2024. Explaining code with a purpose: An integrated approach for developing code comprehension and prompting skills. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 283–289.

[8] 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*. 1163–1169.

[9] 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*. 69–80.

[10] Max Fowler, David H Smith, Mohammed Hassan, Seth Poulsen, Matthew West, and Craig Zilles. 2022. Reevaluating the relationship between explaining, tracing, and writing skills in CS1 in a replication study. *Computer Science Education* 32, 3 (2022), 355–383.

[11] Diana Franklin, Paul Denny, David A. Gonzalez-Maldonado, and Minh Tran. 2025. *Generative AI in Computer Science Education: Challenges and Opportunities*. Cambridge University Press.

[12] Chris Kerslake, Paul Denny, David H Smith, James Prather, Juho Leinonen, Andrew Luxton-Reilly, and Stephen MacNeil. 2024. Integrating Natural Language Prompting Tasks in Introductory Programming Courses. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1*. 88–94.

[13] Viraj Kumar. 2021. Refute: An Alternative to 'Explain in Plain English' Questions. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA) *(ICER 2021)*. Association for Computing Machinery, New York, NY, USA, 438–440. https://doi.org/10.1145/3446871.3469791

[14] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin* 41, 3 (2009), 161–165.

[15] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L Whalley, and Christine Prasad. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin* 38, 3 (2006), 118–122.

[16] 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*. 101–112.

[17] James Prather, Paul Denny, Juho Leinonen, Brett A Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, et al. 2023. The robots are here: Navigating the generative ai revolution in computing education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*. 108–159.

[18] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, Simone Opel, Vee Pettit, Leo Porter, et al. 2024. Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. *arXiv preprint arXiv:2412.14732* (2024).

[19] James Prather, Brent N Reeves, Paul Denny, Juho Leinonen, Stephen MacNeil, Andrew Luxton-Reilly, João Orvalho, Amin Alipour, Ali Alfageeh, Thezyrie Amarouche, Bailey Kimmel, Jared Wright, Musa Blake, and Gweneth Barbre. 2025. Breaking the Programming Language Barrier: Multilingual Prompting to Empower Non-Native English Learners. In *Proceedings of the 27th Australasian Computing Education Conference (ACE '25)*. Association for Computing Machinery, New York, NY, USA, 74–84. https://doi.org/10.1145/3716640.3716649

[20] Brent N Reeves, James Prather, Paul Denny, Juho Leinonen, Stephen MacNeil, Brett A Becker, and Andrew Luxton-Reilly. 2024. Prompts First, Finally. *arXiv preprint arXiv:2407.09231* (2024).

[21] Judy Sheard, Angela Carbone, Raymond Lister, Beth Simon, Errol Thompson, and Jacqueline L. Whalley. 2008. Going SOLO to assess novice programmers. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (Madrid, Spain) *(ITiCSE '08)*. Association for Computing Machinery, New York, NY, USA, 209–213. https://doi.org/10.1145/1384271.1384328

[22] David H Smith, Paul Denny, and Max Fowler. 2024. Prompting for comprehension: Exploring the intersection of explain in plain english questions and prompt writing. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 39–50.

[23] David H Smith and Craig Zilles. 2024. Code Generation Based Grading: Evaluating an Auto-grading Mechanism for" Explain-in-Plain-English" Questions. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 171–177.

[24] David H Smith IV, Viraj Kumar, and Paul Denny. 2024. Explain in Plain Language Questions with Indic Languages: Drawbacks, Affordances, and Opportunities. In *Annual ACM India Compute Conference*. Springer, 3–17.

[25] Annapurna Vadaparty, Daniel Zingaro, David H Smith, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. CS1-LLM: Integrating LLMs into CS1 Instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 297–303.

[26] Matthew West, Geoffrey L Herman, and Craig Zilles. 2015. Prairielearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning. In *2015 ASEE Annual Conference & Exposition*. 26–1238.

[27] Benjamin Xie, Dastyni Loksa, Greg L Nelson, Matthew J Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education* 29, 2-3 (2019), 205–253.

[28] Craig B Zilles, Matthew West, Geoffrey L Herman, and Timothy Bretl. 2019. Every University Should Have a Computer-Based Testing Facility.. In *CSEDU (1)*. 414–420.