Novice programmers inaccurately monitor the quality of their work and their peers' work in an introductory computer science course

Elizabeth B. Cloude*
Faculty of Education and Culture
Tampere University
Tampere, Finland
elizabeth.cloude@tuni.fi

Pranshu Kumar
Department of Computer Science
University of Pennsylvania
Philadelphia, PA, USA
pranshuk@seas.upenn.edu

Ryan S. Baker
Penn Center for Learning Analytics
University of Pennsylvania
Philadelphia, PA, USA
ryanshaunbaker@gmail.com

Eric Fouh
Department of Computer Science
University of Pennsylvania
Philadelphia, PA, USA
efouh@cis.upenn.edu

ABSTRACT

A student's ability to accurately evaluate the quality of their work holds significant implications for their self-regulated learning and problem-solving proficiency in introductory programming. A widespread cognitive bias that frequently impedes accurate self- assessment is overconfidence, which often stems from a misjudgment of contextual and task-related cues, including students' judgment of their peers' competencies. Little research has explored the role of overconfidence on novice programmers' ability to accurately monitor their own work in comparison to their peers' work and its impact on performance in introductory programming courses. The present study examined whether novice programmers exhibited a common cognitive bias called the "hard-easy effect", where students believe their work is better than their peers on easier tasks (overplace) but worse than their peers on harder tasks (underplace). Results showed a reversal of the hard-easy effect, where novices tended to overplace themselves on harder tasks, yet underplace themselves on easier ones. Remarkably, underplacers performed better on an exam compared to overplacers. These findings advance our understanding of relationships between the hard-easy effect, monitoring accuracy across multiple tasks, and grades within introductory programming. Implications of this study can be used to guide instructional decision making and design to improve novices' metacognitive awareness and performance in introductory programming courses.

CCS CONCEPTS

• **Applied computing → Computer-assisted instruction**; **Psychology**; **Education**.

KEYWORDS

Metacognition, Overconfidence, Hard-easy Effect, CS1

ACM Reference Format:
Elizabeth B. Cloude, Pranshu Kumar, Ryan S. Baker, and Eric Fouh. 2024. Novice programmers inaccurately monitor the quality of their work and their peers' work in an introductory computer science course. In *The 14th Learning Analytics and Knowledge Conference (LAK '24), March 18–22, 2024, Kyoto, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3636555.3636848

1 INTRODUCTION

Most novice programmers lack the metacognitive knowledge and skills to harness the benefits of self-regulated learning needed to solve open-ended programming problems [17, 19, 28, 30]. Metacognition is the ability to monitor and regulate one's own cognition to achieve a goal. Numerous studies in computer science education have developed support tools to enhance metacognition among novice programmers, often utilizing automated feedback tools [4]. Despite these efforts, many studies report limited success in using automated feedback tools to improve metacognition, leading to recommendations for explicit instruction from educators [4, 10, 18, 21, 29].

It is possible that novices may not yet possess the level of self-awareness needed to develop their metacognitive abilities. Questions remain about whether automated feedback tools can foster the self-awareness needed to for novices to develop and utilize metacognitive knowledge and skills. Novice programmers require an accumulation of metacognitive experiences to develop self-awarness. By consciously observing cognitive and affective processes (e.g., metacognitive feelings), such experiences can foster valuable insights into one's knowledge, abilities, and limitations while solving problems [26]. As a result, metacognitive experiences play a crucial role in shaping metacognitive knowledge and skills and must be accounted for in the future design and implementation of automated feedback tools in supporting metacognition in introductory computer science (CS) courses.

Challenges exist because every student is susceptible to a distorted level of awareness due to inaccurate self-monitoring. A prevalent bias contributing to this miscalibration is known as overconfidence. Yet, the direction of bias–overconfidence or underconfidence–depends on the conditions of a task and the surrounding context. The hard-easy effect is where this relationship can be observed [16], a common bias where a student underestimates their work on easy tasks but overestimates their work on hard tasks. In contrast, the effect is reversed when a student compares the quality of their work in relation to their peers' work (i.e., placement), where students overplace their work as better than their peers on easy tasks, but worse than their peers on hard tasks. Burson et al. [2] explained that easier tasks tend to produce underestimation and better-than-average effects (i.e., overplacement), while hard tasks tend to produce overestimation, but worse-than-average perceptions of themselves in relation to peers.

To the best of our knowledge, no study in CS education has examined the role of task difficulty on novice program- mers' metacognitive monitoring accuracy. Additional gaps exist as there is little understanding of whether novices' metacognitive monitoring accuracy improves as they progress in the course and may develop more knowledge of programming topics. As such, examining whether metacognitive monitoring accuracy improves over time and across multiple assignments that vary in task difficulty and its relation to performance may provide insights into their learning process and developing metacognitive knowledge and skills. Furthermore, improved metacognitive monitoring accuracy can lead to better self-awareness and meaningful metacognitive experiences, ultimately enhancing learning outcomes and the development of metacognitive knowledge and skills in introductory programming with automated tools.

In this study, we examined whether novices demonstrated the hard-easy effect and the extent to which their metacognitive monitoring accuracy improved across four programming tasks. Next, we examined whether the proportion of metacognitive accuracy or bias across the four programming tasks impacted exam grades in an introductory CS course. The following research questions and hypotheses were investigated:

- **RQ1**: Do novice programmers overplace on easiest homework and underplace on hardest homework? **We hypothesize that novices will overplace on easy tasks and underplace on hard tasks**, as prior studies have found that novices tend to overplace on easy tasks but underplace themselves on hard tasks [23, 27, 32].
- **RQ2**: Do novice programmers' placements change over time, or anchor to the initial homework? **We hypothesize that novices' placement judgments will anchor to the initial homework**, as a prior study found estimation judgments did not change over the course [6]. We expect that students' placement judgments will remain stable and anchor to the initial homework completed, demonstrating an anchoring-and-adjustment heuristic bias [35].
- **RQ3**: Are there differences in performance (exams 1-2) between placement groups? **We hypothesize There will be differences in performance (exams 1-2) grades between novices who more accurately place themselves compared to novices who more inaccurately place themselves.** Based on prior studies [6, 8] and the model of metamemory [25], we expect students who place themselves more accurately will perform better on exams 1-2 compared to novices who place less accurately across multiple homework assignments.

1.1 Studying Metacognition with Automated Tools

In the last two decades, there have been ongoing efforts to study students' metacognition and its relation to performance in CS education [33]. A systematic review Garcia et al. [14] identified several scaffolds and tools designed to measure metacognition while students build their programs. [29] examined novice programmers' metacognition with an automated tool. Novices were randomly assigned to one of two conditions: a prompt and a control. After reading a programming problem prompt, participants in the prompt condition were asked to reflect on a problem statement before writing any code and were directed to a quiz test case. They were asked to calculate the output of the program described in the prompt given a random set of inputs, and once the test case was correct, they could move forward with writing code with an automated feedback tool. In contrast, the control condition could start writing code immediately after reading the problem prompt with an automated feedback tool. Metacognitive processes were collected using a think-aloud protocol in both conditions. The results showed that the prompt condition verbalized significantly more metacognitive processes compared to the control; yet, these differences were only present for those who submitted correct code in the prompt condition. Participants who did not submit correct code demonstrated no differences in metacognitive processes between the conditions. However, the conditions did not differ in performance.

These findings may suggest that, while novice programmers in the prompt condition may have demonstrated more metacognitive processes compared to the control, this difference was only present if the student had submitted correct code. However, regardless of metacognitive differences, it did not impact performance. This result may suggest that the quality of metacognitive processes was low, perhaps due to a lack of self-awareness or accuracy in metacognitively monitoring their work and progress. Questions remain as the authors did not measure how accurate novice programmers' metacognitive monitoring processes were while they built their programs. [20] measured metacognitive monitoring processes with an automated feedback tool by asking participants to self-reflect and explain their programs they built with an automated feedback tool over the course of 10 weeks. The results showed that all of the novices demonstrated at least some degree of metacognitive monitoring and evaluating processes in their reflections and explanations. However, only a few novices reported that they used the insights gained from their reflections and explanations to improve the quality of their work. The analyses also revealed that most novices struggled to reflect and sometimes indicated a complete lack of awareness in their understanding of the programming problem. The authors concluded that novice programmers' metacognition was highly variable and that many may need explicit instruction on how to best monitor their work to improve the quality of programs with automated tools.

Marwan et al. [21] extended this work by building a progress feature into an automated tool that provided feedback on the progress novies made toward a subgoal based on the quality of their code. Using interview and log data to measure metacognitive monitoring processes, novices indicated that the progress feedback supported their metacognitive monitoring processes. However, limitations exist in this study. First, it is unclear whether the progress feedback improved the quality of code. Second, data were not collected on metacognitive monitoring processes used during the programming task, missing critical information on if, when, and how accurate novices' metacognitive monitoring was and its relation to performance.

In sum, many studies show that novice programmers do not initiate accurate metacognitive monitoring, possibly due to a complete lack of self-awareness, and this may impact their performance with automated feedback tools in introductory CS courses. More research is needed to understand factors that may impede novices' ability to accurately monitor the quality of their programs that contributes to better self-awareness and enhances their programming performance with automated feedback tools in introductory CS courses. Further, more research is needed to measure the accuracy of novices' ability to monitor and evaluate their work and its relation to performance with automated feedback tools.

1.2 Metacognitive Accuracy and Performance

While automated tools hold much promise for increasing the quality of programs [14], do automated tools support novices developing the self-awareness needed to utilize metacognition? For metacognition to be effective, novices must accurately monitor and evaluate the quality of their work [8]. Accurate monitoring relies on self-awareness via the novice observing their knowledge and abilities in relation to programming tasks, also known as a metacognitive experience. A metacognitive experience requires the novice to observe how they approach a program and note its failures and successes. To determine how accurate those observations are, they must rely on their metacognitive feelings.

Metacognitive feelings are subjective judgments that involve experiences of confidence, uncertainty, familiarity, difficulty, ease, and comprehension [34]. These judgments provide valuable information about the effectiveness, accuracy, and completeness of cognitive processes and strategies that support self-awareness and metacognitive monitoring. By integrating cues from the environment, task characteristics, and internal cognitive processes, metacognitive feelings emerge and change over time. For example, a novice programmer may feel more confident about their program for an easy task in a small community college course, while their confidence may waver when faced with a difficult task at a prestigious university. Hence, the specific environment and task characteristics contribute to feelings of confidence and influence how one perceives their work. However, environmental and task-related factors can introduce biases that hinder novices from accurately assessing the quality of their work, potentially impacting their performance due to this miscalibration. As a result, it is important to measure the accuracy of metacognitive monitoring via metacognitive feelings such as confidence to determine whether a novice is self-aware while they initiate metacognitive monitoring. Furthermore, understanding the extent and direction of metacognitive monitoring inaccuracy when present may provide valuable insights into factors that contribute to a novice's ability to utilize metacognition effectively with automated tools.

A commonly used method to measure metacognitive accuracy is by comparing feelings of confidence ratings with actual performance [34]. This method evaluates how well-calibrated an individual was in monitoring their work relative to their grade. Traditionally, feelings of confidence have been collected using Likert scales [7, 13]. Moore and Healy [24] operationalize confidence as a multi-dimensional construct, where individuals rely on three judgments: estimation, placement, and precision. Estimation is the most studied facet of confidence in CS education research, and this involves monitoring the quality of one's own work. For example, overestimation indicates the student estimated the quality of their work as better than it was. On the other hand, placement deals with evaluating one's own work by comparing it to their peers' work. Overplacement would describe a student who believed the quality of their work was better than the work of their peers, despite it being lower quality.

The placement dimension of confidence is explained further by social comparison theory [11], where each person is described as having an inherent drive to compare themselves to other people to determine their abilities. Individuals can engage in upward or downward social comparisons, to assess their ability or work relative to peers they perceive as superior or inferior, respectively. A student who overplaces themselves indicates they have a miscalibrated judgment, or inaccurate metacognitive monitoring, where they believe their ability or work surpasses their peers. These comparisons impact a student's feelings of confidence, and this facet is the least studied in CS education. The final facet of confidence is the degree of certainty a student feels about their estimation and/or placement judgment [24].

The accuracy of a metacognitive judgment is determined by the cues a person perceives, which form the basis for their feelings of confidence. Task difficulty impacts the degree of metacognitive bias a student may display. One of the most common biases impacting metacognitive accuracy is known as the hard-easy effect [16]. While evidence of the hard-easy effect is present in a number of research domains, to the best of our knowledge, no study in CS education has directly investigated the role of the hard-easy effect, a common metacognitive bias, on students' ability to accurately place their work across multiple programming tasks. Additional limitations exist as many past studies measure metacognitive accuracy by collecting a single facet of confidence: estimation. This misses information on other facets of confidence, such as placement, that are impacted by the environment and task characteristics. Collecting more information on the components of confidence will aid in identifying if, where, and when metacognitive accuracy and biases emerge over the course of multiple tasks and its impact on performance in introductory programming. Finally, few studies have examined whether novice programmers demonstrate the hard-easy effect and if placement judgments remain stable over time, anchoring and adjusting to the initial cues novices receive about their peers' abilities with automated tools.

1.3 Model of Metamemory

Developing metacognitive knowledge and skills relies on a student accumulating meaningful metacognitive experiences. Metacognitive experiences entail conscious cognitive and affective states that foster self-awareness, metacognitive feelings, and judgments [12, 34]. In this paper, we ground our study in the Nelson and Narens [25] model of metamemory because it serves as a framework for comprehending students' metacognitive processes and memory performance. In this framework, metamemory encompasses the metacognitive processes involved in monitoring and controlling one's memory, including knowledge or strategies employed to enhance performance and metacognitive awareness. Within the model of metamemory, metacognitive accuracy refers to an alignment between students' subjective judgments of their memory performance and their actual objective performance.

Metacognitive accuracy entails two fundamental components: calibration and resolution. Calibration denotes the correspondence between students' subjective judgment in their performance and their actual performance, with high calibration indicating accurate judgments and low calibration signifying metacognitive biases such as overconfidence or underconfidence. Resolution refers to the ability to discriminate between high and low memory performance. A high-resolution metacognitive system enables students to accurately differentiate instances where their memory is likely to be accurate from those likely to be inaccurate. These components of metacognitive accuracy stem from metacognitive feelings which inform the student's metacognitive judgment. For the purposes of this paper, we only focus on how well students' feelings of confidence are calibrated to their actual performance. As previously mentioned [24], students rely on multiple sources of information that contribute to feelings of confidence. Thus, the model of metamemory provides a comprehensive framework for studying the interplay between how well students' feelings of confidence are calibrated to their own and their peers' performance and its impact on exam grades in introductory programming.

1.4 Current Study Objective

The primary goal of this study was to examine the direction and stability of metacognitive accuracy over the course of multiple programming assignments and its impact on exam grades in an introductory programming course. First, we examined if novice programmers' demonstrated a common bias called the hard-easy effect. Next, we examined if novices' placement judgments changed or remained stable over multiple homework assignments, and finally, we assessed whether the direction and proportion of metacognitive inaccuracy over multiple programming assignments impacted exam grades. The findings from this research offer valuable insights for educators in guiding their pedagogical strategies and enhancing student support mechanisms. For instance, by capturing the evolution of novices' accuracy in metacognitive monitoring across a series of programming tasks, educators can gain a deeper understanding of their students' metacognitive biases. This knowledge can then be strategically leveraged to tailor instructional materials and approaches that address these biases, potentially leading to favorable performance outcomes.

2 DATA COLLECTION METHODS

2.1 Participants and Materials

Data were collected from undergraduate students enrolled in an in-person introductory computer science (CS1) course (based in Java) across multiple semesters (Fall 2021, Spring 2022, and Fall 2022) at a large private university in the Northeastern USA. An institutional review board approved this study before data were collected, and the university prohibited the release of demographic information of the students and so these data are missing from our sample. However, for students to enroll in the introductory CS1 course, they had to be at least 18 years old, had not yet declared a major, and demonstrated little experience in programming. At the beginning of the course, the syllabus provided information to students on how each homework assignment would be graded. Students were asked to report their estimation and placement judgments before they submitted their code for each homework assignment to the automated feedback tool. First, students were asked to estimate the grade they would receive on a specific assignment and then they were asked to place the class' average homework grade for the same assignment, yielding two types of judgments per assignment. Since students could resubmit their code to the autograder tool as many times as they wished prior to the deadline, they were prompted to report their judgments every time their code was submitted. The specific questions are below:

- **Estimation judgment:** What grade do you think you will get on this assignment (over 100)?
- **Placement judgment:** What do you think the class average will be on this assignment (over 100)?

2.2 Course Design and Programming Assignments

In the course, students were required to complete 9 programming homework assignments. Students had the option to drop one homework grade from their final course grade. Most students dropped their HW 8 grade; thus, we did not include it in our analysis. The duration of the homeworks ranged from 7 to 14 days before the deadline. Students utilized an online platform called Codio[1], where they completed the programming tasks and had access to an automated feedback tool (i.e., autograder). For each homework assignment, students had unlimited attempts to submit their programs to receive immediate feedback from the autograder, such as error messages. Once code was submitted, the autograder provided students with their scores immediately, regardless of whether estimation or placement judgments were provided. However, students could not view the class' average homework grade until after each homework was due. Codio also provided course resources, including hyperlinks to lecture notes, an interactive electronic textbook that covered programming topics, and various learning activities related to the course topics (e.g., practice problems on recursion or abstract data types). Students also had access to PDF slides on the course topics, which were uploaded to the learner management system by the instructor on a weekly basis.

Students were required to complete eleven quizzes and two examinations throughout the course. Specifically, the examinations

[1]https://www.codio.com/

were administered online and were open-note and open-internet during Fall 2021 (due to COVID-19), allowing students to use the resources they needed to solve the problems, but the examinations and quizzes were closed-note/internet during Spring and Fall 2022. The first exam was administered in the middle of the semester, after the completion of the fourth homework assignment. The second exam was administered at the end of the course after the eighth homework assignment was completed. Table 1 illustrates which programming topics corresponded to each homework assignment and exam.

2.3 Data Coding and Scoring

2.3.1 Homework Grades. The online grading platform Gradescope[2] was used to automatically grade students' programs. We relied on the results obtained from Gradescope to assess the students' grades on each homework assignment, except for homework 8 which was not included in our analysis, as previously mentioned. To facilitate a fair comparison among students, we created an average of each homework grade, in cases where a student submitted their code multiple times to the autograder.

2.3.2 Placement Groups. Students were categorized into one of three groups: overplacers, underplacers, or accurate placers. To do that, we computed JP_i, the projected (or judgment) percentile rank based on the estimate and placement judgment responses (guesses), and AP_i, the actual percentile rank based on actual grades and class average.

Since students were allowed to submit as many times as they wished prior to the deadline per HW, many submitted multiple estimation and placement judgments per homework. To deal with this, we averaged estimation and placement judgments for each HW. We also averaged homework grades of students who submitted their program multiple times. We determined the deviation between the actual percentile (AP_i) and judgment percentile (JP_i) for each student and for each homework assignment, and then classified students as overplacers, underplacers, or accurate placers.

Overplacers ($JP_i>AP_i$) believed they did better than their peers' average grades, while underplacers ($JP_i<AP_i$) believed they did worse than their peers' average grades. To account for slight deviations between AP_i and JP_i, we created an error margin of ±2% around the AP_i, and students who fell within this margin were considered accurate placers.

3 RESULTS

3.1 Preliminary Analysis

To control for the potential impact of selection bias, as students were not required to report their placement judgments in the study, we examined whether students who reported judgments differed in their grades (homework, exams) than those who did not report judgments. A series of Wilcoxon rank sum tests were calculated since our data did not meet normality distribution requirements. The results showed students who reported placement judgments did not differ in performance from those who did not report placement judgments (*ps*>.05). While a marginal effect was observed for homework 4 (p=.0988), its magnitude was very small (<1 point) and did not hold under the post-hoc method. See Table 2 for results. It is important to note that the sample size for exams 1 and 2 under the 'report' column indicates the highest sample at the student level, that is students who provided the placement judgments for the highest number of homework assignments and also had grades for both exams 1 and 2. For example, while homework 1 yielded 237 placement judgments, many of these students did not provide placement judgments for homework 0, 1, and 2. In addition, some of these students dropped the course and thus did not complete the first or second exam. As such, 98 students denotes the highest sample with the most placement judgments, specifically for homeworks 0-3.

3.2 Do novice programmers overplace on the easiest homework and underplace on the hardest homework (hard-easy effect)?

3.2.1 Methods. To answer RQ1, first, the difficulty of each homework assignment was calculated using a two-parameter (2PL) Item Response Theory model (IRT) [5]. IRT is a common approach to estimating the difficulty of assignments for students with varying levels of knowledge. It also provides discrimination estimates to represent how well an assignment distinguishes from another, based on students' knowledge levels. A high discrimination estimate suggests the probability a student did the assignment correctly changed significantly based on the student's level of knowledge. In contrast, the difficulty estimates of the IRT model indicate the knowledge level at which a student has a 50% probability of getting the assignment correct. A higher value corresponds to a more difficult assignment.

To calculate the 2PL IRT model, we used the full sample of students, totaling 711 (n=711). All students were included in this analysis based on whether they received a grade for any of the homework assignments: 0-5 and 7. Homework 6 was not included in our analysis because this assignment changed across the three semesters during data collection. In addition, homework 8 was not included in our analysis (as previously mentioned) because most students dropped the assignment from their final course grade. Similarly they were not able to utilize the automated feedback tool like the other assignments.

To define a passing grade, each grade on the homework assignment was dichotomized based on whether the student received an 'A' letter grade. A '1' was assigned to grades at or above 93 (the A grade cutoff), and the rest were assigned a '0'. This threshold was used because there was low variability in homework scores, likely due to the unlimited submissions students could utilize to iteratively correct and resubmit their code based on the feedback obtained from the automated feedback tool. Afterward, we used the 'lmt' package in RStudio to perform the IRT analysis [31].

Once the difficulty estimates were obtained, we created the easiest and hardest assignments. Next, McNemar's tests were calculated to examine if the frequency of underplacers and overplacers changed from the easiest to hardest assignments, testing the hard-easy effect. McNemar's tests were used because they are designed to analyze differences in the frequencies between paired groups [22]. Accurate placers were not included in our analysis because zero students accurately placed on the hardest homework. In addition,

[2]https://gradescope.com/

Table 1: Homework assignments, programming topics, and exams.

Homework	Topics	Topics	Exams
0	Hello World, Java Syntax, Print statements	30	Exam 1
1	Conditionals and Loops, Variables and Primitive Types	51	Exam 1
2	Functions and String Manipulation	51	Exam 1
3	Functions and Array Manipulation	50	Exam 1
4	Recursion	50	Exam 1
5	Object-Oriented Programming, Unit Testing	50	Exam 2 (Cumulative)
7	Linked Nodes, Unit Testing	52	Exam 2 (Cumulative)

Table 2: Results of Wilcoxon rank sum tests to examine differences between groups.

	Test Statistic		Reported		Did not report	
Homework	*W*	*p*	*n*	*Mdn*	*n*	*Mdn*
0	61544	.7536	333	100	372	100
1	50230	.3516	237	94.12	443	94.12
2	41805	.4402	179	93.14	486	94.12
3	17782	.7263	98	92.31	386	92.31
4	32218	.0988	113	96.08	519	95.1
5	21560	.694	77	97	545	97
7	15820	.7432	58	93.27	560	92.31
Exam 1	28093	.3154	98	82.73	539	80.45
Exam 2	28362	.1219	98	82.05	527	76.82

Note. Mdn=Median grade.

we were more interested in whether there were changes in the direction of metacognitive inaccuracy (e.g., hard-easy effect) across the homeworks.

3.2.2 *Results.* Results indicated that the homework assignments in the course were not extremely difficult (demonstrated by negative values in Table 3). The easiest assignment was homework 0 since it had the lowest difficulty value, while the hardest assignment was homework 3 since it had the highest difficulty value. Refer to Table 3 for detailed model results [3]. These IRT results were used to define the easiest and hardest assignments in the course.

A McNemar's test revealed significant changes in the frequency of underplacers and overplacers from the easiest and hardest homeworks, $\chi^2(1)$=9.0312, p=.002654. Specifically, there were more underplacers on the easiest homework and more overplacers on the hardest homework. These results did not support **Hypothesis 1**, where we expected students to overplace more on the easiest homework and underplace more on hardest homework. In this finding, the hard-easy effect was reversed for the placement groups based on the level of difficulty.

3.3 Do students' placements change over time, or anchor to the initial homework?

3.3.1 *Methods.* To examine if placement judgments changed, or were anchored to the initial homework, placement judgments made on the first 4 homeworks were examined. Specifically, we selected the first 4 homeworks because they included the hardest and easiest assignments and 2 additional assignments that fell somewhere in between. Extracting the placement judgments students reported over the course of the 4 homeworks would allow us to investigate the degree to which placement judgments changed or anchored to the initial homework. An important note to make is that students did not have any information on their peers on the initial homework – homework 0. As such, their placement judgments may change once they have more information on their peers' grades following the initial homework. As previously mentioned, the class average was posted after each homework deadline on the course learning management system, so every enrolled student could access the class's average grade on homework 0 while working on homework 1, but could not do so for homework 0.

Students who reported a placement judgment for the first 4 homeworks were included in our analysis, totaling a sample of 98 students. Placement groups were created based on the placement groups described in the Data Collection Methods section. Using the count of placement groups for the first 4 homeworks, a series of three McNemar's tests were calculated using a Benjamini and Hochberg (B-H) false discovery rate post hoc correction, at a 5% cut-off, to control for multiple testing errors Benjamini and Hochberg [1]. This allowed us to examine whether there were significant changes in the frequency of overplacers, underplacers, and accurate placers from homeworks 0-3.

3.3.2 *Results.* The first McNemar's test revealed marginally significant changes in the frequency of overplacers and underplacers from homework 0 to 1, $\chi^2(1)$=61.253, adj. α=.01667, p<.0001. This finding

[3]Std. errors and z-values in Table 3 correspond to difficulty scores.

Table 3: Median midterm grades across placement groups by each homework assignment.

Homeworks	Difficulty	Discrimination	Std. Error	Z-values
0	**-3.6001**	**1.2426**	**.8507**	**-4.2317**
1	-.5810	1.0162	.1152	-5.0425
2	-.3621	.8145	.1214	-2.9838
3	**.9792**	**1.1654**	**.1123**	**4.3744**
4	-.5794	1.2756	.1044	-5.5490
5	-.8794	1.5380	.1162	-5.0824
7	.1368	1.2671	.0833	-7.567

suggested that the frequency of underplacers and overplacers significantly changed from homework 0 to 1. Specifically, there were significantly more underplacers on homework 0 and significantly more overplacers on homework 1 (Figure 1). In contrast, there were no changes in the frequency of underplacers and overplacers from homeworks 1 to 2 (χ^2(1)=.0062, p=.9372) and homeworks 2 to 3 (χ^2(1)=.4156, p=.5191). This result suggested that the frequency of underplacers was significantly higher than overplacers on homework 0, but the frequency of underplacers was significantly lower than overplacers on homework 1. In addition, there were no significant changes in the frequency of overplacers and underplacers on the rest of the homeworks. This result supported **Hypothesis 2**, where we expected students to adjust and anchor their placement judgments to the initial homework, after receiving information on the class' average homework grade. This suggested that students' placement judgments were possibly adjusted and anchored to the class's average grade on homework 0.

3.4 Are there differences in performance (exams 1-2) grades between placement groups?

3.4.1 Methods. To examine whether exams 1-2 grades differed between placement groups, placement groups were created using the group assignment described in Section 2.3.2. Specifically, students who reported a placement judgment for the first 4 homeworks were included in our analysis, totaling a sample of 98 students. We did not include the other homeworks (5 and 7) in our analysis, since a large portion of students did not report placement judgments on homeworks 5-7. As noted in Table 2, only 58 students reported placement judgment for the last homework, which would not yield a large enough sample to allow for statistical analysis. The sample included the first 4 homeworks since it yielded the largest sample of students who also had exam 1-2 grades.

Based on how accurately students placed their peers in the class, we counted the number of times each student accurately placed, overplaced, or underplaced on the first four homeworks. Next, each student received three separate scores that were based on the percentage of accurate placement, overplacement, and underplacement across the four homeworks. After creating the groups, there were zero students who accurately placed on more than 1 of the 4 homeworks, and because of this, we did not analyze students in terms of their proportion of accurate placement. In addition, we examined whether the direction of metacognitive accuracy had an impact on exam grades.

To determine if there were differences in exam grades between the over/underplacement groups, we examined if our data met the assumptions needed to calculate an Analysis of Variance (ANOVA). Two Shapiro-Wilks tests were calculated to examine if our data met assumptions of normality. The results showed significance for each response variable: exam 1 (W =.91312, p<.0001) and exam 2 grades (W=.89516, p<.0001). As a result, two separate Kruskal-Wallis tests were calculated to examine differences in the rank of exam 1-2 grades between overplacers and underplacers The Kruskal-Wallis test is the non-parametric alternative to the one-way ANOVA, but does not assume data hold a normal distribution. If significant differences in grades were detected between over/under placement groups, a pairwise Dunn's test was calculated to determine exactly which group was different [9].

3.4.2 Results. A Kruskal-Wallis test revealed significant differences in exam 2 grades between overplacement groups, χ^2(3)=6.108, adj. α=.0333, p=.0333. A pairwise Dunn's test revealed significant differences in exam 2 grades, where students who overplaced on 25% of the homeworks scored significantly higher (Mdn=90.0) than students who overplaced on 50% of the homeworks (Mdn=74.1), Z=2.7501, adj. α=.0083, p=.0059. In contrast, there were no differences in exam 2 grades between students who overplaced on 75% of the homeworks (Mdn=74.1) and 100% of the homeworks (Mdn=70). Surprisingly, there were no differences in exam 1 grades between any of the overplacement groups (ps>.05; Table 4).

The second Kruskal-Wallis test found significant differences in exam 2 grades between underplacement groups, χ^2(3)=11.336, adj. α=.01667, p=.01004. A pairwise Dunn's test found significant differences in exam 2 grades, where students who underplaced on 25% of the homeworks scored significantly lower on exam 2 (Mdn=73.4) than students who underplaced on 100% of the homeworks (Mdn=93.2), Z=-2.6883, adj. α=.0083, p=.0072. Interestingly, there were no differences in exam 2 grades between students who underplaced on 50% (Mdn=82.5) of the homeworks and 75% of the homeworks (Mdn=85.5). Similar, there were no differences in exam 1 grades between underplacement groups (ps>.05).

Findings showed that very few students accurately placed on more than 1 of the 4 homeworks, showing that accurate metacognition was rare. Notably, students who overplaced on 1 of the 4 homeworks (25%) had significantly higher exam 2 grades than those who overplaced on 2 of the 4 homeworks (50%). In contrast, those who underplaced on all of the homeworks (100%) scored significantly higher on exam 2 than those who underplaced on 1 of the 4 homeworks (25%).

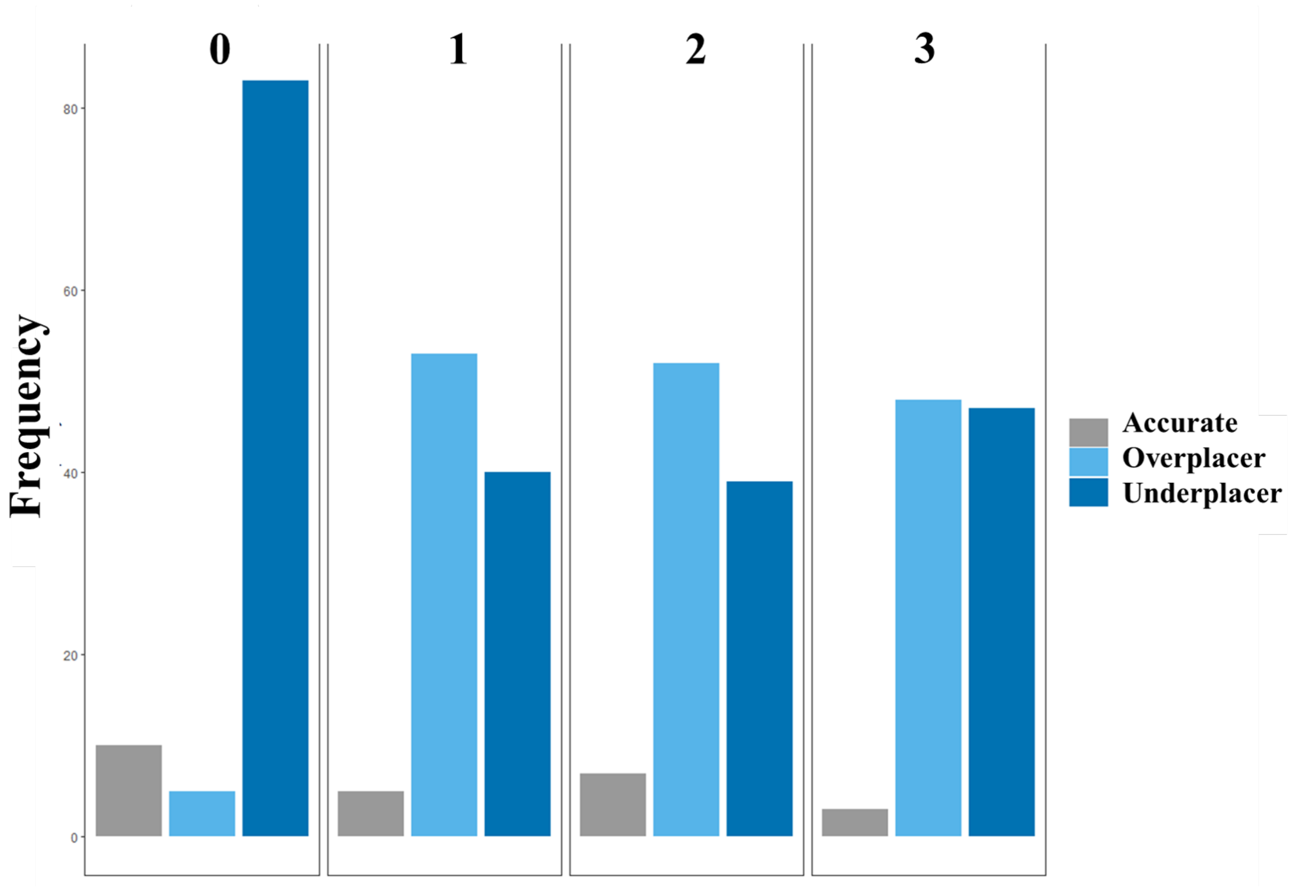

Figure 1: Distribution of placement groups across homeworks 0-3.

These results do not support **Hypothesis 3**, where we expected differences in exam 1-2 grades between the placement groups, specifically higher exam grades for accurate placers compared to inaccurate placers. While we found that underplacers performed significantly better on exam 2 than overplacers, there were no differences in grades between accurate and inaccurate groups. Similarly, there were no differences in exam 1 grades between placement groups. A possible explanation for this result could be due to the fact that most of the students in the course did not accurately place himself regardless of the homework assignment task, indicating a lack of metacognitive awareness altogether. The lack of accurate metacognitive monitoring altogether could explain why there were no differences in exam grades between accurate and inaccurate placement groups.

Interestingly, those who underplaced more often on the homework assignments than overplaced, had significantly higher exam 2 grades. This result was surprising, as we did not expect miscalibrated metacognitive monitoring to benefit performance. A possible explanation for this could be due to the fact that students who believed the quality of their work was lower than their peers may have sought out more help, instruction, and other resources to further develop their knowledge and skills, possibly contributing to higher exam 2 grades as a result. Since these students believe that their quality of work is lower, they may be more motivated or put forth more effort to improve the quality of their work. In contrast, students who overplaced, or believed their work was better than their peers, may not have put forth as much effort as underplacers, since they did not suspect that the quality of their work was better than average, possibly contributing to significantly lower exam 2 grades.

4 DISCUSSION

The primary goal of this study was to examine the direction and stability of metacognitive monitoring accuracy across multiple programming assignments and its impact on exam grades in an introductory programming course. In **RQ1**, we examined whether novices demonstrated the hard-easy effect in their placement judgments. The analyses indicated that there were significantly more underplacers on the easiest homework than overplacers. In contrast, there were more overplacers on the hardest homework than underplacers. These results did not support **Hypothesis1**, where we expected students to overplace more on the easiest homework and underplace more on hardest homework [16], as found by prior studies [23, 27, 32]. A possible explanation for these findings could be due to the task-related cues students received on the easy and hard homeworks. For example, the hard-easy effect explains that a

Table 4: Median exam grades between placement groups.

	Placement Groups					
	Underplacer			Overplacer		
% of placement direction across 4 homeworks	*n*	Mdn_1	Mdn_2	*n*	Mdn_1	Mdn_2
25%	**64**	80.9	**73.4**	**17**	86.8	**90.9**
50%	18	87.5	82.5	19	87.3	82.3
75%	13	85	85.5	**59**	80.9	**74.1**
100%	**3**	88.2	**93.2**	3	74.5	70

Note. Mdn_1=Median exam 1 grade; Mdn_2=Median exam 2 grade; **Boldface** indicates p<.05.

student is likely to underplace the quality of their work when they perceive task-related cues that indicate the task is hard rather than easy. In contrast, students tend to adhere to better-than-average beliefs about themselves when they perceive task-related cues that indicate the task is easy [2]. Thus, the reversed effect may stem from task-related cues that corresponded with the easy and hard tasks. Many of the novices who enrolled in the introductory programming course had little to no exposure to programming tasks. Thus, students' knowledge of task-related cues that indicate varying levels of difficulty would not have been developed by the initial homework (easy). In addition, the students have very little information on their peers and their abilities. In contrast, the hardest homework was the fourth assignment which was assigned during the middle of the course. Students may have begun developing knowledge of task-related cues by completing the other programming assignments and may have learned more about their peers' backgrounds, experiences, and abilities.

RQ2 examined whether novices' placement judgments changed or anchored to the initial homework assignment. The results revealed significant changes in the number of overplacers and underplacers from the initial homework to the second homework in the course. However, there were no changes in the number of overplacers and underplacers for the remaining homeworks. This result indicated that students underplaced themselves more on the first homework, yet overplaced themselves more on the next three homeworks. This result supported **Hypothesis 2**, where we expected students to adjust and anchor their placement judgments to the initial homework, after they received more information about their peers' abilities. Once the first homework was graded, the class average grade on was publically available via the learner management system. It is likely students utilized this information as an indicator of the quality of their peers' work for the next homeworks. These results also support Denny et al. [6], which found that the accuracy of students' metacognitive judgments did not improve as they progressed through a programming course. This result may indicate that novices' metacognitive calibration anchors to the initial task and remains stable across multiple tasks with varying difficulty. These results contradict findings from [3], where the accuracy of judgments improved over time and over tasks. However, a notable distinction between [6] and [3] from our study, is that confidence was measured using estimation judgments instead of placement judgments. Thus, the findings may stem from different task-related cues students relied on to generate estimation and placement judgments.

Finally, **RQ3** examined whether novices who more accurately placed themselves on the four homeworks performed significantly higher on exams 1 and 2 compared to those who inaccurately placed. The analyses showed, first, very few students accurately placed themselves on more than 1 of the 4 homeworks. It is important to note that the degree of inaccurate metacognitive calibration was determined based on how much deviation allowed between the judgment and objective score. We allowed students an error margin of 2-5%, such that students who were close to an accurate judgment were not penalized. Thus, the threshold that determines "accurate enough" is a limitation in this research and future researchers may need to explore what is "accurate enough", even though the student is not perfectly accurate in their placement judgment. The results also showed that students who overplaced on less homeworks had higher exam 2 grades than students who overplaced more. Similarly, the more often students underplaced, the higher their exam 2 grades were compared to students who underplaced on less homeworks. Surprisingly, there were no differences in exam 1 grades between any of the placement groups. These findings did not support **Hypothesis 3**, where we expected differences in performance between those who were more accurate in their placement judgments compared to inaccurate judgments. Rather, the direction of inaccurate calibration – underplacement vs overplacement – impacted performance differently. Underplacers performed better than Overplacers, and this was contradictory to the model of metamemory [25] and prior studies [6, 8] which suggest that inaccurate metacognition is detrimental to performance. However, similar results were identified by [15], which found that feelings of underconfidence were beneficial to performance compared to overconfidence.

A possible explanation for the lack of differences in exam 1 grades between accurate and inaccurate placement groups could be the fact that exam 1 was designed to evaluate the first few programming topics covered in the course. Exam 2 was cumulative and covered all the topics in the course (Table 1). Thus, grades on exam 1 may not holistically represent the knowledge that students had developed in the course, whereas exam 2 grades may better represent the knowledge developed. To explain why underplacers performed better than overplacers on exam 2, this result could stem from a belief that students who underplace hold; they believe their work needs improvement and may have sought out help and resources. If a student believed their work was of lower quality than their classmates' work, they may have asked the instructor for help, possibly resulting in higher exam 2 grades. In contrast, students who overplaced believed their work was better than their classmates, and thus may not have seen value in seeking help or improving their work, possibly contributing to lower exam 2 grades.

Overall, our findings reveal that novice programmers demonstrated metacognitive monitoring bias on multiple programming assignments, regardless of task difficulty. While this did not result in exam 1 grade differences, students who underplaced performed significantly better on exam 2 than overplacers. We also found that novices' demonstrated a reversed hard-easy effect, where they underplaced more on the easiest assignment and overplaced more on the hardest assignment. Finally, we found evidence that novices' placement judgments anchored and adjusted to the initial homework assignment, demonstrating no improvement in metacognitive accuracy over the course.

4.1 Threats to Validity

This was an in-person course and participation was voluntary; thus, some students did not provide placement judgments for some assignments. As a result, the sample may not represent all students who completed the course. However, students did not differ in performance between those who reported and those who did not (Table 1). Another limitation involves using IRT to define assignment difficulty. Due to the low variability in homework grades, these data may not best represent how difficult an assignment was (grades only reflect the final product of the programs). However, the iterative submissions may have promoted metacognitive monitoring by using the feedback obtained from the automated feedback tool, possibly improving programs.

4.2 Future Directions and Implications

Future research should consider extending this line of work by operationalizing feelings of confidence as a multidimensional construct that encompasses three components: estimation, placement, and precision [24]. In addition, distinguishing between the level of metacognitive judgment (local vs. global) and the type of cues each judgment may rely on may explain factors of a task and environment that introduce bias and impede novice programmers from accurate metacognitive monitoring. We also recommend future researchers investigate whether novices demonstrate other common biases that contribute to inaccurate metacognitive monitoring and its role on programming performance. Examining the role of biases on other metacognitive processes and self-regulated learning, such as whether underplacers engage in help-seeking behaviors more than overplacers, may explain why sometimes inaccurate metacognitive monitoring can also be beneficial for performance.

Implications of this research may advance our understanding of factors (e.g., task-related cues) that can best support novices' developing self-awareness and metacognition in introductory programming with automated feedback tools. In addition, providing instructors with information on their students' placement judgment accuracy across programming assignments could contribute to integrating instructional materials and pedagogical strategies to improve students' metacognitive awareness. For example, identifying and understanding the metacognitive biases that students hold in introductory programming courses can support instructors in implementing targeted instructional interventions to improve students' learning and performance. This shift can be used to guide instructional design and pedagogy, rooted in empirical evidence and metacognitive theory, which may yield significant benefits in improving novices' metacognitive skills and knowledge and performance outcomes in introductory programming.

ACKNOWLEDGMENTS

This study was supported by the National Science Foundation (NSF; DUE-1946150). Any conclusions expressed in this material do not necessarily reflect the views of NSF. We would also like to thank other project members, Nigel Newby, Maciej Pankiewicz, and Valdemar Švábenský, for their valuable feedback and insights on the preliminary results. ChatGPT 4 was also used to assist in editing the text and building scripts in RStudio.

REFERENCES

[1] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)* 57, 1 (1995), 289–300.

[2] Katherine A Burson, Richard P Larrick, and Joshua Klayman. 2006. Skilled or unskilled, but still unaware of it: how perceptions of difficulty drive miscalibration in relative comparisons. *Journal of personality and social psychology* 90, 1 (2006), 60.

[3] Huanyi Chen and Paul A. S. Ward. 2022. Metacognitive Accuracy in Homework Assignments, Time-Limited Quizzes, and Learning Objectives. In *Proceedings of the 30th International Conference on Computers in Education. Asia-Pacific Society for Computers in Education.*

[4] Michelle Craig, Andrew Petersen, and Jennifer Campbell. 2019. Answering the correct question. In *Proceedings of the ACM Conference on Global Computing Education.* 72–77.

[5] Rafael Jaime De Ayala. 2013. *The theory and practice of item response theory.* Guilford Publications.

[6] Paul Denny, Andrew Luxton-Reilly, John Hamer, Dana B Dahlstrom, and Helen C Purchase. 2010. Self-predicted and actual performance in an introductory programming course. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education.* 118–122.

[7] John Dunlosky and Janet Metcalfe. 2008. *Metacognition.* Sage Publications.

[8] John Dunlosky and Katherine A Rawson. 2012. Overconfidence produces underachievement: Inaccurate self evaluations undermine students' learning and retention. *Learning and Instruction* 22, 4 (2012), 271–280.

[9] Olive Jean Dunn. 1964. Multiple comparisons using rank sums. *Technometrics* 6, 3 (1964), 241–252.

[10] Katrina Falkner, Rebecca Vivian, and Nickolas JG Falkner. 2014. Identifying computer science self-regulated learning strategies. In *Proceedings of the 2014 conference on Innovation & technology in computer science education.* 291–296.

[11] Leon Festinger. 1954. A theory of social comparison processes. *Human relations* 7, 2 (1954), 117–140.

[12] John H Flavell. 1979. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist* 34, 10 (1979), 906.

[13] Stephen M Fleming and Hakwan C Lau. 2014. How to measure metacognition. *Frontiers in human neuroscience* 8 (2014), 443.

[14] Rita Garcia, Katrina Falkner, and Rebecca Vivian. 2018. Systematic literature review: Self-Regulated Learning strategies using e-learning tools for Computer Science. *Computers & Education* 123 (2018), 150–163.

[15] Brian Harrington, Shichong Peng, Xiaomeng Jin, and Minhaz Khan. 2018. Gender, confidence, and mark prediction in CS examinations. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education.* 230–235.

[16] Sarah Lichtenstein and Baruch Fischhoff. 1977. Do those who know more also know more about how much they know? *Organizational behavior and human performance* 20, 2 (1977), 159–183.

[17] Dastyni Loksa and Amy J Ko. 2016. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM conference on international computing education research.* 83–91.

[18] Dastyni Loksa, Amy J Ko, Will Jernigan, Alannah Oleson, Christopher J Mendez, and Margaret M Burnett. 2016. Programming, problem solving, and self-awareness: Effects of explicit guidance. In *Proceedings of the 2016 CHI conference on human factors in computing systems.* 1449–1461.

[19] Dastyni Loksa, Lauren Margulieux, Brett A Becker, Michelle Craig, Paul Denny, Raymond Pettit, and James Prather. 2022. Metacognition and self-regulation in programming education: Theories and exemplars of use. *ACM Transactions on Computing Education (TOCE)* 22, 4 (2022), 1–31.

[20] Dastyni Loksa, Benjamin Xie, Harrison Kwik, and Amy J Ko. 2020. Investigating novices' in situ reflections on their programming process. In *Proceedings of the 51st ACM technical symposium on computer science education*. 149–155.

[21] Samiha Marwan, Preya Shabrina, Alex Milliken, Ian Menezes, Veronica Catete, Thomas W Price, and Tiffany Barnes. 2021. Promoting students' progress-monitoring behavior during block-based programming. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research*. 1–10.

[22] Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 12, 2 (1947), 153–157.

[23] Edgar C Merkle. 2009. The disutility of the hard-easy effect in choice confidence. *Psychonomic bulletin & review* 16 (2009), 204–213.

[24] Don A Moore and Paul J Healy. 2008. The trouble with overconfidence. *Psychological review* 115, 2 (2008), 502.

[25] Thomas O Nelson and Louis Narens. 1994. Why investigate metacognition. *Metacognition: Knowing about knowing* 13 (1994), 1–25.

[26] Paul R Pintrich. 2002. The role of metacognitive knowledge in learning, teaching, and assessing. *Theory into practice* 41, 4 (2002), 219–225.

[27] TJ Pleskac and J Busemeyer. 2010. Two-stage dynamic signal detection: A theory of confidence, choice, and response time. *Psychological Review* 117, 3 (2010), 864–901.

[28] James Prather, Brett A Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What do we think we think we are doing? Metacognition and self-regulation in programming. In *Proceedings of the 2020 ACM conference on international computing education research*. 2–13.

[29] James Prather, Raymond Pettit, Brett A Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM technical symposium on computer science education*. 531–537.

[30] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive difficulties faced by novice programmers in automated assessment tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. 41–50.

[31] Dimitris Rizopoulos. 2006. ltm: An R package for Latent Variable Modelling and Item Response Theory Analyses. *Journal of Statistical Software* 17, 5 (2006), 1–25. https://doi.org/10.18637/jss.v017.i05

[32] James D Sauer, Matthew A Palmer, and Neil Brewer. 2019. Pitfalls in using eyewitness confidence to diagnose the accuracy of an individual identification decision. *Psychology, Public Policy, and Law* 25, 3 (2019), 147.

[33] Leonardo Silva, António José Mendes, Anabela Gomes, and Gabriel Fortes Cavalcanti de Macêdo. 2021. Regulation of learning interventions in programming education: A systematic literature review and guideline proposition. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 647–653.

[34] Pina Tarricone. 2011. *The taxonomy of metacognition*. Psychology press.

[35] Amos Tversky and Daniel Kahneman. 1974. Judgment under Uncertainty: Heuristics and Biases: Biases in judgments reveal some heuristics of thinking under uncertainty. *science* 185, 4157 (1974), 1124–1131.