Promoting Students' Progress-Monitoring Behavior during Block-Based Programming

Samiha Marwan
samarwan@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Preya Shabrina
pshabri@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Alexandra Milliken
aamillik@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Ian Menezes
ivmeneze@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Veronica Catete
vmcatete@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Thomas W. Price
twprice@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Tiffany Barnes
tmbarnes@ncsu.edu
North Carolina State University
Raleigh, NC, USA

ABSTRACT

Providing students with adaptive feedback on their progress on programming problems has been shown to motivate students and improve their performance, but little is known about how such feedback might impact student self-regulated learning during programming. Self-regulated learning (SRL) involves student planning a task, monitoring their progress, and reflecting on the outcome. We explored students' SRL behaviors, particularly progress monitoring, when programming using each of three different scaffolds. The first scaffold is a subgoal checklist for the given programming task, the second adds automated, binary completion feedback on each subgoal, and the third adaptively reflects an automated percent progress estimate of student progress on each. Through interviews and programming logs from 17 students solving a problem in a block-based programming environment, we investigated the extent to which each scaffold supported student SRL. Our qualitative study results suggest that all three scaffolds could be useful for student SRL, but students felt that a combination of the checklist and progress feedback provided them with a balance of autonomy and motivation to persevere in programming. Furthermore, our results suggest that explaining how the automated feedback system works may have encouraged students to reason about the feedback they receive, which was a key intended outcome to improve SRL during programming.

KEYWORDS

self-regulated learning, block-based programming, progress monitoring, progress feedback

ACM Reference Format:
Samiha Marwan, Preya Shabrina, Alexandra Milliken, Ian Menezes, Veronica Catete, Thomas W. Price, and Tiffany Barnes. 2021. Promoting Students' Progress-Monitoring Behavior during Block-Based Programming. In *21st Koli Calling International Conference on Computing Education Research (Koli Calling '21), November 18–21, 2021, Joensuu, Finland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3488042.3488064

1 INTRODUCTION

Learning to program can be quite challenging, especially for novice learners, and these challenges can lead to negative self-assessments [15]. In these situations, learners benefit from encouragement and feedback on their progress, helping them to persevere when they are capable of solving a problem but unsure about their ability to do so [8, 31]. This progress feedback can come from an instructor, but recent work has shown that students also benefit from *automated* progress feedback [17, 26]. This feedback is *immediate and adaptive*, highlighting students' progress, as well as their mistakes, in real-time as they work, and it has been shown to increase students' performance [12, 29], and intentions to persist in Computer Science (CS) [26]. This goes beyond traditional "autograders," which are generally designed to assess a *complete* program. Rather, feedback on students' progress can explore the potential for automated feedback when it is needed most: immediately, during problem solving when students are not yet ready to submit for auto-grading [9, 26]. Such feedback may be particularly impactful in block-based and novice-centered learning environments, as it may help guide students' natural exploration and "tinkering" towards correct solutions [10].

However, while *automated* progress feedback provides critical support to students during problem solving, it also raises questions about *how* such feedback may affect students' *own* ability to monitor their progress – a key self-regulatory skill that is important for learning [22]. If students can rely on automated progress feedback, will they trust it blindly and stop verifying their own work? Or, alternatively, could such feedback encourage students to reflect *more often* on how they are progressing with respect to assignment goals? This question is particularly important for *automated feedback*, since it lacks the expert judgement of a human and can be misleading [36, 38, 44]. Given the potentially faulty guidance, it is important that students reflect on their own programs' correctness and monitor their own progress. The answer to the question of how students' self-regulated learning is impacted by automated progress feedback may also depend on *how* such support is presented to the students. To investigate this question, we investigated how three different scaffolds could support students in monitoring their programming progress in a block-based programming environment. Our first scaffold is a **checklist** of subgoals for the given programming task, which students can check to self-monitor their progress. Our second scaffold provides the same list but with adaptive, immediate **completion** feedback that indicates when each subgoal is complete during problem solving. Our third scaffold provides adaptive immediate **progress** feedback, that indicates how much progress a student has made toward completing each subgoal.

In this paper we asked the following research questions, **RQ1: How do students enact progress monitoring when given scaffolds (i.e. subgoal checklist, progress feedback) during block-based programming?**, and **RQ2: What are students' perceptions of each scaffold?** To answer our research questions, we ran a pilot qualitative study with 17 students and randomly assigned them to solve a programming task using one of the three scaffolds: checklist, completion, or progress. For the two adaptive scaffolds (completion and progress), we also verbally communicated how the feedback system works, as a way to encourage students to reason about the feedback, particularly when it may not match student expectations. We analyzed programming trace logs and interviews to investigate our research questions. We found that all three scaffolds led most students to follow the ordered subgoal checklist, and monitor their progress on the subgoals while programming. Our log data analysis also suggests that the adaptive progress feedback interface acted as a reminder for students to monitor their progress more frequently, and to adjust their problem-solving strategy. Our results suggest that, rather than replacing a students' own progress monitoring process, automated progress feedback – and even a non-automated subgoal list – can support students in their own progress monitoring, and encourage them to reflect on the feedback. Both of these key aspects of self-regulated learning behavior are important learning goals for open-ended programming assignments, but may not be explicitly taught or tracked in introductory programming assignments.

2 RELATED WORK

Programming feedback is essential for students' learning and motivation to learn [43]. A growing body of work in computing education has developed and evaluated various computer-based tutors that provide automated programming feedback [18, 19, 35, 45]. Depending on the programming language, such feedback can have various forms, such as compiler error messages [2], hints [27], or assessment feedback (e.g. autograders' feedback) [1, 45]. In our literature review we focus on feedback provided in block-based programming environments.

Block-based programming environments are inherently designed to simplify the programming process for novices, for example by reducing the burden of receiving syntax errors [7, 13]. However, researchers have also integrated them with various automated feedback features to scaffold students' performance and learning during problem-solving [1, 17, 26–28, 35, 45]. For example, Ball used correct instructor solutions to develop an autograder for the Snap*!* block-based programming environment, to help students assess their code. Price et al. extended the Snap*!* block-based programming environment with a system that provides data-driven hints to help novice students when they get stuck [35]. In the "BlockPy" block-based programming environment, Gusukuma et al. developed immediate misconception-based feedback, and found that it improved students' performance [16]. While these support features can improve students' performance by helping them fix mistakes, or suggesting missing components, they have not been evaluated with respect to students' self-regulatory skills, such as planning, or progress monitoring, which play an important role in learning [11, 22], self-efficacy, and growth mindsets [23].

Self-regulated learning (SRL) "is a cognitively and motivationally active approach to learning" [46], where students monitor and modify their learning strategies in response to outcomes. In programming, Loksa et al. proposed a theoretical framework for the role of SRL, including planning, process monitoring, comprehension monitoring, reflection on cognition, and self-explanation [22]. Their think-aloud study with CS1 and CS2 students showed infrequent, inconsistent self-regulation behaviors, and most were shallowly applied, if at all. However, Kumar et al. found that embedding SRL principles in a Java learning environment improved students' performance in programming [21]. In this work, we focus on "progress monitoring" – a key aspect of SRL that is important for a number of reasons. First, progress monitoring helps students stay motivated as they can keep track of their progress, and adapt strategies when they are not making progress, leading to increased progress [14]. Second, progress monitoring also guides students to adjust their help-seeking and reason about the feedback they received, which can improve their learning [6, 25, 27]. Sawyer showed that most learners need support to monitor their progress effectively, and they need feedback to verify their own self-evaluations [39]; however, there is little work that examines how to promote these skills in programming [21, 23]. For example, promoting students to create subgoal labels for a given task can be considered as a form of helping students plan their work, which has been shown to improve their performance [24]. However, it is unclear if creating subgoals can support students' SRL behavior. In addition, students, particularly novices, often make many incorrect code design choices, and it is hard for novices to predict the effectiveness of their design choices [39]. This suggests that students need support, (or scaffolds), to help them develop good SRL skills (e.g. progress monitoring, self-explanation, and self-assessment), which can improve their problem-solving efficiency and retention.

Visual languages and novice progress monitoring. Visual languages are built to provide students with immediate feedback upon running the code for visual tasks, such as moving a sprite. While it may seem obvious how to use this feature to consistently monitor progress, novices often engage in unproductive tinkering behaviors, such as avoiding running/testing code for long periods of time, or repeatedly running their code without making any changes [10]. This suggests that novice students need more scaffolds for the problem-solving process, i.e. through progress feedback. Additionally, for programming tasks with many valid solutions, it is a challenge to provide perfectly accurate automated progress feedback [34, 42], showing the importance of promoting students' progress monitoring skills so they can verify the received feedback.

3 FEEDBACK SYSTEM AND INTERFACE DESIGN

3.1 Adaptive Feedback System

The basis of our feedback system, embedded in the programming environment shown in Figure 1, relies on cutting-edge technologies to build data-driven autograders for subgoals of a specific programming task, called Subgoals Detectors [26]. To implement these detectors, two experts first constructed a list of subgoals of a given programming task based on how prior students have approached this task. We then implemented a data-driven autograder to detect the completion of each subgoal, while students are programming, by comparing their current code with prior students' correct solutions. We call these autograders subgoal detectors, described in detail in our prior work [26]. In prior work, we embedded our subgoal detectors in Snap*!* [13], a block-based programming environment, for simple programming tasks (with solutions of 7-10 code blocks), to provide students with adaptive feedback on whether they completed each subgoal. In a camp study, we found that this adaptive feedback increased students' persistence in CS, and improved their performance on future tasks [26]. In a classroom study, we found that the system improved students' engagement with a homework programming task [29]. We also investigated interactions between feedback and student behavior - showing that accurate feedback during problem solving could reduce student idle time, but students responded differently to the completion feedback, sometimes ignoring it, or sometimes following system feedback, without running their code to self-assess or reflect on it [42].

In this work, we used the same feedback system, but for an open-ended programming task (described in Section 4). In addition, we extended each subgoal detector to calculate progress based on the percentage overlap between students' current code and the code features used by prior students to complete that subgoal. In this work, we explore *how* different interfaces with different scaffolding levels, that rely on the subgoals detectors, can support students and promote their progress monitoring while they program an open-ended programming task, as we discuss below.

3.2 Three Interface Designs

In our prior work, we found that high school students working in the Snap*!* programming environment needed assistance to plan their work and keep track of their progress [26]. Further analysis of this data (not reported in the publication) showed that students used a variety of unorganized problem-solving strategies that led to inefficient performance. These findings prompted us to investigate the impact scaffolds may have on student progress monitoring. Based on learning research, breaking a task into subgoals facilitates learning in programming [24], and providing this list within the programming environment satisfies the multimedia learning design principle of **contiguity** - spatially placing assistive content where it is needed [32].

Our first interface is a **checklist** of hand-crafted labels of subgoals as shown in Figure 1. The checklist interface does not provide adaptive feedback, but allows students to indicate when they feel each subgoal is complete. This checklist gives us the opportunity to investigate students' planning and progress monitoring on the task, and provides a *baseline* for comparison with the other two adaptive feedback interfaces.

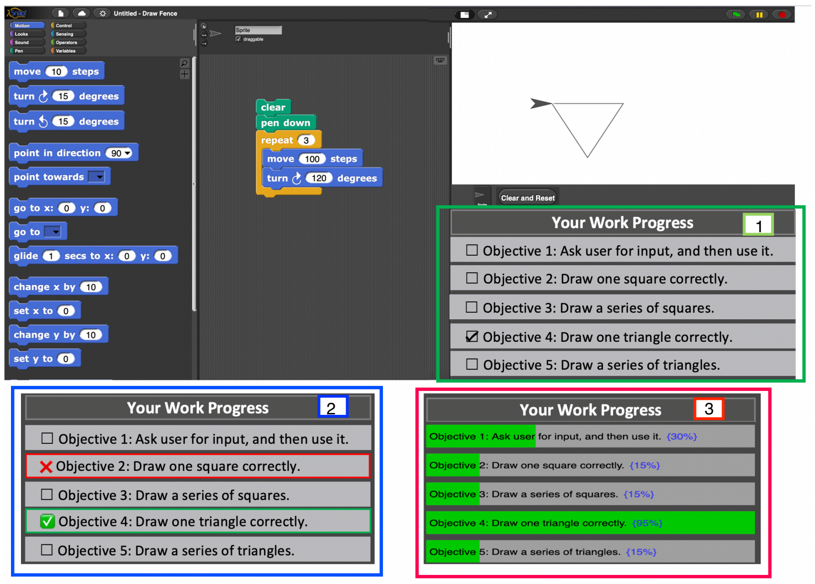
We designed the second two interfaces to provide adaptive feedback, but each with a different level of scaffold, guided by learning theories on effective forms of feedback [40, 43], to support student progress monitoring. *Interface 2* provides adaptive **completion feedback** by adding colored highlights to the subgoal list, as shown at the bottom left of Figure 1, in green when a subgoal is *complete*, or red when a subgoal is subsequently *broken*, *during* problem solving. This interface follows Scheeler et al.'s effective performance feedback design principles to be (1) immediate (detected in real time), (2) specific (highlighting the relevant subgoal), (3) positive (confirming completion), and (4) corrective (indicating mistakes) [40]. *Interface 3* provides adaptive **progress feedback** using a visual progress bar and numeric estimate of percent completion for each subgoal, as shown at the bottom right of Figure 1. Using progress bars for each subgoal is similar to skill meters, which is the most common representation for progress monitoring used in learning environments [4, 30]. Providing continuous progress feedback has been shown to be important for student motivation, particularly when students cannot determine progress on their own [41]. We also allow the interface to update the progress bar for each subgoal when it passes intervals of about 15% so the changes would be noticeable, but not distracting.

In summary, in this work we took design principles from prior work on limitations and theories of effective feedback [40, 43], and incorporated them as features in a block-based programming environment. As discussed below, we conducted a qualitative pilot study to explore how these interfaces with different levels of scaffolds can support student progress monitoring behavior.

4 METHODS

The goal of this study is to answer the following research questions: *How do students enact progress monitoring when given scaffolds (i.e. subgoal list and adaptive progress feedback) during block-based programming?*, and *What are students' perceptions of each interface?* To answer these questions, we conducted a qualitative user study with 17 students solving a programming task using one of the 3 interface designs described in Section 3.2, followed by student interviews. We used both log data and interview data to investigate students' progress monitoring behavior, and their perceptions on the interfaces, as detailed below.

Participants: We recruited participants for this IRB-approved study from a summer high school computer science internship program. Each intern had some prior programming experience, with 1 or 2 Computer Science courses and 5 weeks in the block-based coding internship. Internship advisors invited participants



Your Work Progress 1

- ☐ Objective 1: Ask user for input, and then use it.
- ☐ Objective 2: Draw one square correctly.
- ☐ Objective 3: Draw a series of squares.
- ☑ Objective 4: Draw one triangle correctly.
- ☐ Objective 5: Draw a series of triangles.

Your Work Progress 2

- ☐ Objective 1: Ask user for input, and then use it.
- ❌ Objective 2: Draw one square correctly.
- ☐ Objective 3: Draw a series of squares.
- ✅ Objective 4: Draw one triangle correctly.
- ☐ Objective 5: Draw a series of triangles.

Your Work Progress 3

- Objective 1: Ask user for input, and then use it. {30%}
- Objective 2: Draw one square correctly. {15%}
- Objective 3: Draw a series of squares. {15%}
- Objective 4: Draw one triangle correctly. {95%}
- Objective 5: Draw a series of triangles. {15%}

Figure 1: Snap*!* programming environment, where feedback was placed on the bottom right of the interface according to student condition. *Interface* 1 (green outline, inset) has checkboxes, *Interface* 2 (blue, bottom-left) has highlights and system checks, and *Interface* 3 (red, bottom-right) has green progress bars with percent progress.

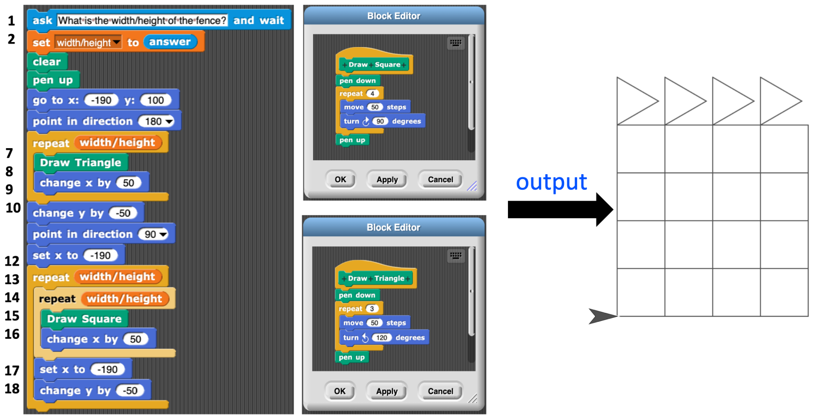
to the study verbally and via Slack. Seventeen interns participated, aged 15-18, with 14 females and 3 males.

Procedure: The first author, a researcher in CS education, conducted a scripted one-on-one study with each participant[1]. First, the participant took a pre-survey to collect consent, demographics, and prior programming background. Next, the researcher asked students to read instructions for the *Draw Fence* task and complete it in Snap*!*. Figure 2 shows one students' correct solution to the *Draw Fence* task and its output. This program takes user input for the width and height of a picket fence composed of a grid of squares with triangles on the top. Based on the instructions given with the *Draw Fence* programming task, we divided the task into 5 subgoals, created a subgoal detector for each, then augmented the Snap*!* programming environment with a text describing each subgoal. As shown in Figure 2, subgoal 1 requires the student to ask for user input for the fence height and width (lines 1-2, Figure 2). Subgoal 2 requires students to draw a square as shown in the 'Draw a Square' custom block (similar to a function). Subgoal 3 requires drawing a series of squares (lines 13-18, Figure 2), where the 'Draw Square' custom block was nested in a 'repeat' block (i.e. a loop) that iterates 'width/height' times. Subgoal 4 requires drawing a triangle which was implemented using the custom block 'Draw Triangle'. Subgoal 5 requires drawing a series of triangles (lines 7-9, Figure 2). The *Draw Fence* task can be solved using several strategies that correspond to distinct sets of subgoals.

Next, each student was assigned randomly to one of three interface conditions (described in detail above in Section 3). *Condition* 1 (n=6), *Condition* 2 (n=5) and *Condition* 3 (n = 6) correspond to *Interface* 1, 2, and 3, respectively. For *Condition* 1, the researcher noted that the system provides a list of suggested subgoals, and students have the option to check or uncheck any subgoal when they feel it was complete or not. In *Conditions* 2 and 3, the researcher introduced the system with the following statement: **"This is a new system for assessment feedback, and it's based on matching your code to what previous students have done to solve the same problem. Since everyone codes differently, it's not always 100% correct".** This was done to promote student reflection and verification about the adaptive feedback, particularly when it does not match their expectations. In *Conditions* 2 and 3, while students were programming, our adaptive feedback system (i.e. subgoals' detectors) continuously and simultaneously determined progress and completion for each subgoal. For example, in Figure 1, Condition 2, student changes resulted in simultaneous updates to the completion feedback for multiple subgoals, showing that the student broke subgoal 2 (i.e. drawing a square) while attempting a later subgoal (i.e. drawing a triangle).

After the student completed the task, or 30 minutes, whichever came first, we interviewed each student for 10-15 minutes, to investigate how the interface was helpful, and to what degree the student was relying or reflecting on the system feedback. In particular, the researcher asked about (1) whether the feedback system was helpful or unhelpful, and why, and (2) what information students wanted to know during programming, (from a human or automated feedback). At the end of the interview, the researcher demonstrated the other two conditions (i.e. interfaces), and asked for students' preferences and design suggestions.

[1]Due to the COVID-19 pandemic, we conducted this study online via Zoom.



```
1   ask [What is the width/height of the fence?] and wait
2   set [width/height] to (answer)
    clear
    pen up
    go to x: (-190) y: (100)
    point in direction (180)
    repeat (width/height)
7     Draw Triangle
8     change x by (50)
9
10  change y by (-50)
    point in direction (90)
    set x to (-190)
12  repeat (width/height)
13    repeat (width/height)
14      Draw Square
15      change x by (50)
16
17    set x to (-190)
18    change y by (-50)
```

```
Draw Square
pen down
repeat (4)
  move (50) steps
  turn ↻ (90) degrees
pen up
```

```
Draw Triangle
pen down
repeat (3)
  move (50) steps
  turn ↺ (120) degrees
pen up
```

Figure 2: Draw Fence Solution with line numbers on left. In the middle are two custom blocks (i.e. functions), one to draw a square (top) and one to draw a triangle (bottom), and the script's output is on the right.

Log Data Analysis: Snap*!* programming environment logs all student code edits while programming (e.g. adding or deleting a block), as a code trace, as well as the feedback system actions (e.g. detecting the completion of a subgoal). This logging feature allows us to detect all student steps, feedback system detections, as well as the *time* taken for each step. We used students' log data to (1) grade student work manually, (2) inspect the accuracy of the systems' feedback detections in *Condition* 2 and 3, and (3) analyze students' code traces to illustrate student progress monitoring behaviors with each interface. We also used log data to calculate a number of measures of how students interacted with the scaffolds. Based on themes that emerged in this aggregate analysis, we selected case studies to present – some of which reflect the general trends in the data, and some of which show less common student outcomes. We used this log data analysis to answer RQ1 as explained in detail in Section 5.

Interview Data Analysis: We transcribed the 17 interview recordings containing 204 minutes of audio. Analysis was done by four co-author coders, a team consisting of one professor and three computer science graduate students, each with 2-4 years experience in conducting and analyzing data from related user studies.

Our qualitative analysis occurred in three rounds as in [33]: inductive/open coding, and 2 rounds of deductive qualitative coding. In the first round, we divided the first two interviews into uninterrupted student utterances, usually consisting of 1-2 sentences (often called "segments"). Next, we independently analyzed the segments and collaborated to resolve conflicts and define a codebook. In round 2, we used the resulting codebook to tag the next two interviews, discussing codes and resolving conflicts in 6/108 (5%) segments, updating the codebook with a total of 180 codes. In round 3, two coders performed independent deductive coding on the remaining 13 interviews, consulting the remaining team as needed. Finally, one coder combined frequent codes that appear in at least 5 segments into a set of themes: how each interface was helpful/unhelpful, students' preferences, and suggestions for improvement. We then related these themes to each interview question. We report these themes with example quotes from students, in Section 5.4, to present students' perceptions about each interface. In our results, when reporting quotes from interview data we report participants' ID preceding their quote (e.g. [P1] means participant 1).

5 RESULTS

We organize our results around our two research questions. We first analyze log data to investigate RQ1: how students enacted progress monitoring when given a subgoal list (Section 5.1), adaptive feedback (Section 5.2), and even when they encountered unexpected feedback (Section 5.3). We then use the interview data to address RQ2, investigating students' perceptions, in Section 5.4.

5.1 Providing Students a List of Subgoals

Our first method of investigating how students might have enacted progress monitoring is by measuring the extent to which students *followed* the subgoals presented by the list in each of the 3 interfaces, and to what extent they used this list for progress monitoring. Our log data analysis showed that 16 out of the 17 students completed the subgoals in the exact same order as presented, and 12 of these students completed the programming task. Additionally, in *Condition* 1, where students did not receive adaptive feedback, our log data analysis showed that 5/6 students were checking off subgoals when they decided they were complete. Together, these results suggest that presenting students with a subgoal list may help in promoting students' progress monitoring by giving students a clear plan they *can* monitor their progress towards, and an aid for keeping track of that progress.

In addition, our log data analysis also suggests that such progress monitoring may lead to improved performance. We found that the 16 students, who completed subgoals in order, spent an average of 14.5 minutes until they stopped working on the task. This is in contrast to what we found in prior work with other highschool students working on the same task [26], where students, without access to the subgoal list, solved the problem in a variety of unorganized ways, spending at least double the time spent by students in this study. While this was a different population, the extreme differences in time, coupled with the different order of subgoals completed, suggest the subgoal list may have benefitted students. Furthermore, when we investigated log data of the one student who did not follow the subgoal list (who was assigned to *Condition* 1), we found that they solved the problem in a complicated approach, spending 421 code edits in ∼ 30 minutes, which is the maximum time spent by all students in our study. When the interviewer asked this student if they think the *checklist* is helpful, the student said: *"I jumped in there. I didn't plan it out. I didn't do any of that - I was just like, Oh, this shouldn't be too hard at all, and I'm midway, I just realized I made a lot of mistakes. I should have spent more time planning than writing my code."* While this may suggest that embedding the subgoal list in Snap*!* was not enough to capture this student's attention, these results show that providing students with a subgoal list (i.e. a plan) may lead to improved performance as well.

Findings: These results suggest that providing students with a subgoal list, even a non-adaptive one, seemed to be a helpful scaffold that promoted students' progress monitoring behavior. This is evidenced by the fact that students followed the order of subgoals in the list, showing that they paid attention to it, and also that students checked off subgoals as they worked – a form of progress monitoring. While we cannot say whether our participants would do similar progress monitoring without the scaffolds, analyzing student code traces from our prior study (as discussed above) suggest that they do not follow a common process. Unlike the program of the single student who ignored the subgoal list, those programs written by the rest of students who followed the subgoal list were more organized, understandable, and completed faster, showing improved performance. These students felt the list *"makes things a lot easier cuz it sort of like laid out how you should do the program. So instead of having to jump right in and come up with your own, like method of doing it, you can just look at the [subgoals]"* [P1].

5.2 Subgoal List with Completion and Progress Feedback

One of our goals with *Conditions* 2 and 3 was to scaffold students' progress monitoring by alerting them when their detected progress changed, to encourage students to reflect on their progress at these moments. Therefore, our second method of exploring how students might have enacted progress monitoring is by investigating to what extent students responded to progress feedback, and how they reacted, in both cases of positive feedback (completed subgoals) and negative feedback (lost progress on a subgoal). In our log data analysis, in *Condition* 2 and *Condition* 3, we noted that when students completed a subgoal which was detected by our feedback system, 10/11 students always moved on and started working on the subsequent subgoal. We also noted that when the feedback reported a *decrease* in progress after a student made an edit, most students responded to the feedback by undoing the edit. This decrease in progress was presented in *Condition* 2 and *Condition* 3 as broken subgoals highlighted in red or decreased subgoal progress percent, respectively. Specifically, under *Condition* 2, we found 7 cases when a subgoal was broken while a student was solving the *Draw Fence* task. In 6 of those 7 cases, students brought back the removed blocks (that caused a broken subgoal) within 1-3 steps. In *Condition* 3, we found 22 cases when the progress of a subgoal decreased by some amount, which is notably higher than that in *Condition* 2 because the progress feedback was more sensitive to every edit, and not every decreased instance corresponded to breaking an entire subgoal. In 15 of those cases (68.18%), the students *reverted* the changes causing the decrease in progress within 1-4 steps. In the other seven cases, the students ignored the detections. Six of those cases occurred for the same student, Beth[2]. Ignoring the feedback might indicate limited progress monitoring behavior, which can lead to decreased performance. We present here a case study of Beth, to explore how and why they ignored the feedback, and the possible consequences of such behavior.

Case Study Beth: Ignoring decreased progress in Condition 3. Beth started their attempt by completing subgoal 1 and added an 'ask' block to take user input for width/height of the fence to be drawn. They added 'move', 'turn', and 'pen down' statements which are required for multiple subgoals (e.g. to draw a square, or a triangle). Beth then removed the 'move' and 'turn' statements, decreasing subgoal 2 progress from 20% to 10%. They ignored or did not notice this decrement, and did not bring the statements back until making 17 additional unhelpful edits that did not show any progress. Then, they brought back the 'move' and 'turn' blocks and nested them in a 'repeat' block making progress towards subgoals 2 and 4. In the subsequent step, Beth removed the 'move' and 'turn' statements from within the 'repeat' block, decreasing the progress of subgoals 2 and 4 from 65% to 35%, but this time they immediately brought these blocks back. Later, Beth made several unhelpful edits, repositioning code blocks that variably caused the progress of subgoals 2 and 4 to decrease by 10%, and 80%, respectively. There were a total of 10 cases of progress decreases found in Beth's code logs, but Beth only reverted the changes immediately in 4 of these cases. In the other 6 cases, Beth did not revert the changes at all or reverted after a significant number of edits. Beth failed to complete the task within the allocated time.

During the interview, when we asked Beth what was helpful in the feedback, they said: *"I think it's really helpful. The only thing that kind of threw me off a bit is when I started, [the subgoal list] shows progress in the triangles' [subgoal] when I hadn't started coding the triangles yet. I was just a little confused"*. This clarifies why Beth might have decided to ignore the progress feedback. This is interesting since it presents a case where the feedback was correct but the student thought it was not. For context, using 'pen down' and 'move' blocks are needed to draw a triangle and that is why the system showed increased progress in subgoal 4; however, since the student was working on subgoal 2, this led to confusion about the system. When the interviewer reminded Beth of how the system worked and why it detected an increase in subgoal 4, Beth said: *"I feel like I would definitely trust it. I think that maybe something that would have been a little helpful for me is knowing what you said about how it calculates like the pen down. I think if I had known that information before, I wouldn't have been confused at all when I was doing the programming."*

Findings: These results show that most students receiving feedback on subgoals mirrored that feedback (in both conditions), suggesting that they were using the feedback to monitor their progress. This is evidenced by the fact that students often fixed their edits that led to decreased progress. Furthermore, in the interviews, *all*

[2]We provide students with random names for anonymity.

students in *Condition* 3 mentioned that the incremental progress feedback was helpful in letting them see their progress (though this was not reported by 2/5 students in *Condition* 2, which we discuss in Section 5.4). For example, [P7] said *"it helps students see that they are actually making progress even when they think that they're not, like if they're totally stuck, just placing some blocks and then see that Oh, I'm actually making progress towards this."* However, Beth's case shows that a lack of attention towards the feedback or frequently ignoring the feedback may decrease its usefulness, which bolsters suggestions in prior work [28, 37].

5.3 Students' Role in Progress Monitoring

Our third method of exploring how students might have enacted progress monitoring is by investigating whether students have engaged in progress monitoring and reflection at moments when the automated feedback system provided slightly early or late detection of subgoals. To perform this analysis, three co-author experts collaboratively assessed student log data to detect precisely when subgoals were completed. The three experts labeled 55 subgoals (from 11 students in *Conditions* 2 and 3 with 5 subgoals each), as ***on-time*** (54.3%) if they agreed with the system completion detection timing, ***early*** (9%), if the system detected the completion of a subgoal slightly *earlier* than the expert (i.e. one or two edits before the experts thought it was complete), or ***late*** (36.7%) otherwise. Next, we analyzed student log data and documented student responses to the slightly *early* and *late* detections to determine whether student behaviors reflected their engagement in appropriate *progress monitoring*.

In the 5 *early* detection cases in *Conditions* 2 and 3, none of the student programs were working as intended. In all of these cases students continued working until they achieved the desired output, suggesting that the early detection feedback *did not* mislead these students to believe that they had completed the subgoal. In 7 of the 9 *late* detection cases in *Condition* 2, and 10 of the 11 *late* detection cases in *Condition* 3, students moved on to the next subgoal, deciding to override the *late* detection. These results suggest that *late* detections, which comprised 36% of *Condition* 2 and 37.9% of *Condition* 3 detections, only potentially negatively impacted 3 students, who continued working on the same subgoals even after a *late* detection. We illustrate the extent of potential negative progress assessment on case study Kim, the student in *Condition* 3 who continued working on a subgoal that was already complete but was detected *late* by the feedback system.

Case Study Kim: Impact of late detections while programming. Kim started working on subgoal 1. Before completing it, expert analysis shows that Kim completed subgoals 2, 3, 4, and 5 in order, in 9 minutes and 14 seconds. At this point, only subgoal 1 was incomplete (the user input was not used in the program); but the feedback system showed 64% progress on subgoal 1, and *erroneously* showed 55% progress on subgoal 5. Kim spent 5 minutes and 51 seconds more making edits related to subgoal 5. Finally, Kim made one edit to complete subgoal 1. However, subgoal 5 remained undetected by the system until the end of Kim's attempt.

During the interview with Kim, when we asked their opinion about the feedback they received, they said: *"it was really useful because it was like step by step, so you really know like what to work on first."* We then asked Kim how the feedback was helpful or unhelpful in this task, and they said: *"with the like percentages. I knew how much I was progressing, like if I'm completely blocked. For this particular program, I think drawing a series of triangles (i.e. subgoal 5) didn't really incorporate in my program as much as like, since that's why I didn't like finish it."* It is clear from Kim's response that even after they started the interview they still thought that they did not complete subgoal 5, although it was complete. When the interviewer reminded them how the feedback system works, Kim said: *"I think maybe [it is] a little misleading. Once I actually completed [subgoal 5], and I saw it was 55%, it made me want to just figure out [what my code was missing] even though like I could build the fence in the code was working properly."* The interviewer then followed up by asking for suggestions on how we could better communicate how the system works to students, and Kim recommended: *"I guess we can just clarify it [in] an info button that explains what these subgoals do, and [that] if it's not fully completed, it's fine."* Kim's response suggests that we could remind students how the feedback system works within the programming environment.

Findings: Based on students' code logs and this case study, we conclude that students generally do not blindly follow feedback, suggesting that feedback does prevent student progress monitoring. Both log data and interview data show that most students assess their progress when the system shows that it has changed, and overrule the system if they disagree. This is further reflected from the interviews, where students noted using the output of their program, or their own knowledge to monitor their progress and taking control by ignoring feedback that they did not agree with. When asked about this, one student in *Condition* 2 stated "P2: I guess it's just like, other people did it differently or something like that." However, case study Kim cautions us that this feedback system still needs refinement, as some students may need further support to engage in effective progress monitoring.

5.4 Students' Preferences across Interfaces

While the code data analysis and case studies pointed us to *how well* and *when* the subgoals list, with or without adaptive feedback, can promote students' progress monitoring behavior, the interview data complemented this analysis by emphasizing *why* the subgoal lists were useful or not, and collect student preferences and suggestions on how to mitigate these issues.

Through our thematic analysis, we found that the two main design features perceived as useful by students are: breaking down the task with a list of subgoals, and providing them with adaptive progress feedback (not just completion). For example, [P1] stated that the subgoals' list "doesn't tell you the exact steps on how to do it, but it does give you the steps and then you can piece them together, just like, figure out where everything should go." In addition, most students noted how it makes the task easier as it gives them a plan, to tackle the solution, and *guidance* on how to proceed, which satisfies the outcomes of the *contiguity* design principle [32]. For *Condition* 3, all students perceived the progress feedback as useful. For example, [P9] noted that: *"the part where it makes you most motivated is when you're stuck, and then you do something and then you see the percentage rise up and then you're like,* *Oh, yeah, I'm getting somewhere."* It is interesting that presenting a progress interface to students allows adaptive feedback not only to help students to monitor their progress, but also to motivate them when they feel stuck, reflecting the benefits of applying effective feedback design choices [40].

When we asked students about the *unhelpful* aspects across the 3 interfaces, they noted a reoccuring theme of times when they received unexpected feedback. For example, [P11] said: *"it's not always 100% accurate like as you see my program works but it still hasn't checked off the last box, but [I] know why it did so, because maybe [I] did the series of triangles in a different way."* However, it was clear from their responses that students *reasoned* that their solution strategy might be different from prior student solutions. This emphasized that explaining how the system works promoted students' self-regulatory behaviors of reflection and progress monitoring.

When we asked students about which interface they prefer, and why, 14/17 students suggested combining *Condition* 1 (the checklist) and *Condition* 3 (progress feedback). For example [P13] said: *"personally, if I was using it, I guess, I would like it to show my progress, but also give me an option of, like checking it off."* . Additionally, all students noted that they prefer *Condition* 3 over *Condition* 2, since progress feedback made them feel more motivated to complete the task, as in prior studies on progress feedback in the SQL Tutor [30]. Additionally, some students in *Condition* 2 noted that, when they did notice a subgoal being broken (turning red), it was hard to determine which code edits broke the subgoal. This indicates that the *Condition 2*'s completion feedback did not provide enough specific feedback for some students.

6 DISCUSSION

RQ1: How do students enact progress monitoring when given scaffolds during block-based programming? Our qualitative pilot study suggested three main results showing how our scaffolds' designs can promote students' progress monitoring behavior during programming. First, we found evidence that **visualizing a subgoal list within a programming environment, whether it provided adaptive feedback or not, provided a plan that students followed, and used to monitor their progress throughout their work**. This is important, since prior work suggests that determining how to start, and plan out the work, are two key programming moments where negative self-assessments can occur for novices [15]. Students' interview responses revealed that providing a subgoal list to be helpful during these important moments: [P7] *"it was definitely helpful because the first thing that you think is how do I approach this problem? and what should I start with? Because you're starting with a blank screen. So it was definitely helpful to see like, we've broken up the task into steps that you can follow."* We also found that students used the subgoal list in *planning* as reflected in students' log data, where we found that all students except one worked on the task subgoals in order, i.e. they used the list as a plan.

Second, we found suggestive evidence that **providing a subgoal list with either type of adaptive feedback (completion or progress) promoted students' progress monitoring**. Loksa et al. identified that a key part of students' self-regulation during programming is "to plan and evaluate progress toward writing a program that solves some computational problem [22]." Our log data analysis in Section 5.2 shows that students generally responded to decreased progress feedback immediately (68-85% of the time), probably by redoing the code edits they did before, or by using trial-and-error strategies as suggested in prior work [10]. Regardless, this suggests that the system successfully encouraged students to monitor their progress leading to a higher performance. Additionally, our results in Section 5.3 show that students were not simply relying on the *system's* assessments, but were ultimately monitoring their *own* progress – overriding erroneous feedback. In 17/20 (85%) of *late* detections, students moved on to the next subgoal appropriately, rather than adding code to get higher completion feedback, and no student failed to complete a subgoal that was detected *early*. While Kim's case study shows that some students were still misled, indicating the importance of accurate feedback, our overall results suggest that automated progress feedback supports, rather than replaces, students' own progress monitoring.

Third, we found evidence in our interviews that **explaining how the feedback system works, and why it may not align with student expectations, supported students to reflect on the feedback they received**. While data-driven feedback systems can be overall helpful to students [38, 44], this challenge of generating feedback that does not align with students' expectations is present in many of these feedback systems [36]. In this paper we acknowledged this challenge, and we explored how to mitigate its potential harm by improving students' self-regulation skills of progress monitoring and reflection on the feedback they received. We prompted students to reason about the feedback by verbally explaining that it was based on comparisons of their code to prior student work. While research shows that student help seeking behaviors can be quite sensitive to hint quality [37], our results suggest that explaining how the system worked helped promote students to reason about the feedback. Additionally, all students using *interface* 3 noted that adaptive progress feedback motivated them to complete the programming task, even if some of the received feedback does not match their expectations. These results confirm Bodily et al.'s thoughts that exposing how the system works to students would inspire confidence and learner self-regulation [3].

RQ2: What are students' perceptions of and preferences for such progress monitoring scaffolds? Measuring students' perceptions is important to evaluate the success of any feedback mechanism [20], and specifically important in shaping researchers' design choices of feedback systems in the future. Our interview data analysis showed that most students preferred to have the checkboxes, in *interface* 1, to track their own assessment of their progress along with adaptive progress feedback (i.e. combining *interface* 1 and *interface* 3). This thematic finding reflects learners' need for combining both human intelligence (i.e. deciding if a subgoal is complete) and machine intelligence (i.e. automated detections of students' progress on each subgoal), which is a lacking feature in automated feedback systems. Not only do we think that this combined condition might motivate students to complete programming tasks and mitigate the consequences of receiving unexpected feedback; but also it can improve students' learning as suggested that prompting students to reflect or self-explain what a solution step means, is a constructive activity that engages students in active learning, while allowing them to monitor their understanding [5]. However, since these results are from a qualitative study, large controlled studies are needed to confirm this effect.

7 LIMITATIONS

Like any study, this work has some limitations. First, our study consists of a small population. However, the goal of our pilot study is to explore how different scaffolds can promote students' progress monitoring behavior. Second, the interviews may reflect response bias that may have led to more positive answers. To mitigate potential bias, we asked open-ended questions about both the positive and negative aspects of the feedback design. Third, since the students had a five-week experience in programming, they are likely to possess both higher self-regulation skills and a higher ability to understand when their code might appropriately differ from other student solutions, when compared to students in an introductory programming course. However, our sample was purposefully selected to interview students who would be capable of reflecting on their earlier programming experiences and what might have helped them. During interviews, several students mentioned ways that our feedback systems could be more helpful to beginners, such as providing, on-demand, step-by-step suggestions on how to complete a subgoal. In addition, our prior work evaluated a feedback system that has an interface similar to *interface* 2 with novice students, who showed clear benefits from having completion feedback during programming [26]. Fourth, our study is limited to one programming task. However, the task includes several programming concepts (e.g. loops, variables, and user-input) and requires complex nested structures making it very likely for students to have diverse solutions, which our results confirm since 16 out of 17 students had different solutions. Fifth, our study does not have a control group since our goal from our qualitative study is to explore how interfaces with different scaffolding levels can support students and promote their progress monitoring while they program - not an experimental comparison to show which feedback interface is best. Sixth, and last, while students' gender might have impacted their behavior and perceptions about the interfaces, comparing genders was out of scope from this paper because we had only 3 male participants.

8 CONCLUSIONS

In this paper, we conducted a qualitative study with 17 high-school students to explore how three subgoal-tracking interfaces can promote students' progress monitoring skills during programming: 1- a checklist, with no feedback, 2- completion feedback and 3- progress feedback. We found that breaking down a task into a list of subgoals provided students with a plan that helps them to monitor their progress, and adaptively showing to students their progress (*interface* 3) promoted student progress monitoring. We also found that explaining how the feedback is generated might have helped students to reason about the feedback, and ignore it when it did not match their expectations. Our interview data suggests that it is important to design adaptive learning environments that inform students about their progress, while enabling them to record their own self-assessment that reflects their perceptions of whether a subgoal is complete. In future work, we plan to evaluate an interface that combines *interface* 1 with *interface 3*, and evaluate its impact on students' outcomes in a large-scale study.

REFERENCES

[1] Michael Ball. 2018. *Lambda: An Autograder for snap.* Technical Report. Electrical Engineering and Computer Sciences University of California at Berkeley.

[2] Brett A. Becker, Kyle Goslin, and Graham Glanville. 2018. The Effects of Enhanced Compiler Error Messages on a Syntax Error Debugging Test. (2018).

[3] Robert Bodily, Judy Kay, Vincent Aleven, Ioana Jivet, Dan Davis, Franceska Xhakaj, and Katrien Verbert. 2018. Open learner models and learning analytics dashboards: a systematic review. In *Proceedings of the 8th international conference on learning analytics and knowledge.* 41–50.

[4] Susan Bull, Abdallatif S Abu-Issa, Harpreet Ghag, and Tim Lloyd. 2005. Some Unusual Open Learner Models.. In *AIED.* 104–111.

[5] Michelene TH Chi. 2009. Active-constructive-interactive: A conceptual framework for differentiating learning activities. *Topics in cognitive science* 1, 1 (2009), 73–105.

[6] Lindy Crawford and Leanne R Ketterlin-Geller. 2008. Improving math programming for students at risk: Introduction to the special topic issue. *Remedial and Special Education* 29, 1 (2008), 5–8.

[7] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated transfer: Alice 3 to java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education.* 141–146.

[8] Barbara Di Eugenio, Davide Fossati, Stellan Ohlsson, and David Cosejo. 2009. Towards explaining effective tutorial dialogues. In *Annual Meeting of the Cognitive Science Society.* 1430–1435.

[9] Roberta E Dihoff, Gary M Brosvic, Michael L Epstein, and Michael J Cook. 2004. Provision of feedback during preparation for academic testing: Learning is enhanced by immediate but not delayed feedback. *The Psychological Record* 54, 2 (2004), 207–231.

[10] Yihuan Dong, Samiha Marwan, Veronica Catete, Thomas W. Price, and Tiffany Barnes. 2019. Defining Tinkering Behavior in Open-ended Block-based Programming Assignments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* ACM, 1204–1210.

[11] 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.

[12] Davide Fossati, Barbara Di Eugenio, STELLAN Ohlsson, Christopher Brown, and Lin Chen. 2015. Data driven automatic feedback generation in the iList intelligent tutoring system. *Technology, Instruction, Cognition and Learning* 10, 1 (2015), 5–26.

[13] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The beauty and joy of computing. *ACM Inroads* 6, 4 (2015), 71–79.

[14] Nuno Gil Fonseca, Luís Macedo, and António José Mendes. 2018. Supporting differentiated instruction in programming courses through permanent progress monitoring. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education.* 209–214.

[15] Jamie Gorson and Eleanor O'Rourke. 2020. Why Do CS1 Students Think They're Bad at Programming? Investigating Self-Efficacy and Self-Assessments at Three Universities. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20).* Association for Computing Machinery, New York, NY, USA, 170–181. https://doi.org/10.1145/3372782.3406273

[16] Luke Gusukuma, Austin Cory Bart, Dennis Kafura, and Jeremy Ernst. 2018. Misconception-driven feedback: Results from an experimental study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research.* 160–168.

[17] Luke Gusukuma, Dennis Kafura, and Austin Cory Bart. 2017. Authoring feedback for novice programmers in a block-based language. In *2017 IEEE Blocks and Beyond Workshop (B&B).* IEEE, 37–40.

[18] I-H Hsiao, Sergey Sosnovsky, and Peter Brusilovsky. 2010. Guiding students to the right questions: adaptive navigation support in an E-Learning system for Java programming. *Journal of Computer Assisted Learning* 26, 4 (2010), 270–283.

[19] David E Johnson. 2016. ITCH: Individual Testing of Computer Homework for Scratch Assignments. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education.* ACM, New York, NY, 223–227.

[20] Samad Kardan and Cristina Conati. 2015. Providing adaptive support in an interactive simulation for learning: An experimental evaluation. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.* ACM, New York, NY, 3671–3680.

[21] Vive Kumar, Philip Winne, Allyson Hadwin, John Nesbit, Dianne Jamieson-Noel, Tom Calvert, and Behzad Samin. 2005. Effects of self-regulated learning in programming. In *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05).* IEEE, 383–387.

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

[23] Dastyni Loksa, Andrew 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.

[24] Lauren Margulieux and Richard Catrambone. 2017. Using learners' self-explanations of subgoals to guide initial problem solving in app inventor. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 21–29.

[25] Samiha Marwan, Anay Dombe, and Thomas W. Price. 2020. Unproductive Help-seeking in Programming: What it is and How to Address it?. In *The Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science (ITiCSE'20)*. ACM, New York, NY.

[26] Samiha Marwan, Ge Gao, Susan Fisk, Thomas W. Price, and Tiffany Barnes. 2020. Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science. In *Proceedings of the International Computing Education Research Conference (forthcoming)*.

[27] Samiha Marwan, Joseph Jay Williams, and Thomas W. Price. 2019. An Evaluation of the Impact of Automated Programming Hints on Performance and Learning. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. ACM, 61–70.

[28] Samiha Marwan, Nicholas Lytle, Joseph Jay Williams, and Thomas W. Price. 2019. The Impact of Adding Textual Explanations to Next-step Hints in a Novice Programming Environment. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 520–526.

[29] Samiha Marwan, Thomas W Price, M. Chi, and Tiffany Barnes. 2020. Immediate Data-Driven Positive Feedback Increases Engagement on Programming Homework for Novices. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop @ EDM'20*.

[30] Antonija Mitrovic and Brent Martin. 2007. Evaluating the effect of open student models on self-assessment. *International Journal of Artificial Intelligence in Education* 17, 2 (2007), 121–144.

[31] Antonija Mitrovic, Stellan Ohlsson, and Devon K Barrow. 2013. The effect of positive feedback in a constraint-based intelligent tutoring system. *Computers & Education* 60, 1 (2013), 264–272.

[32] Roxana Moreno and Richard E Mayer. 1999. Cognitive principles of multimedia learning: The role of modality and contiguity. *Journal of educational psychology* 91, 2 (1999), 358.

[33] Alannah Oleson, Meron Solomon, and Amy J Ko. 2020. Computing Students' Learning Difficulties in HCI Education. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, 1–14.

[34] Thomas W Price, Yihuan Dong, and Tiffany Barnes. 2016. Generating Data-Driven Hints for Open-Ended Programming. *International Educational Data Mining Society* (2016).

[35] Thomas W. Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. ACM, New York, NY.

[36] Thomas W Price, Yihuan Dong, Rui Zhi, Benjamin Paaßen, Nicholas Lytle, Veronica Cateté, and Tiffany Barnes. 2019. A comparison of the quality of data-driven programming hint generation algorithms. *International Journal of Artificial Intelligence in Education* 29, 3 (2019), 368–395.

[37] Thomas W. Price, Zhongxiu Liu, Veronica Catete, and Tiffany Barnes. 2017. Factors Influencing Students' Help-Seeking Behavior while Programming with Human and Computer Tutors. In *Proceedings of the International Computing Education Research Conference*. ACM, New York, NY.

[38] Thomas W. Price, Rui Zhi, and Tiffany Barnes. 2017. Hint Generation Under Uncertainty: The Effect of Hint Quality on Help-Seeking Behavior. In *Proceedings of the International Conference on Artificial Intelligence in Education*.

[39] R Keith Sawyer. 2005. *The Cambridge handbook of the learning sciences*. Cambridge University Press.

[40] Mary Catherine Scheeler, Kathy L Ruhl, and James K McAfee. 2004. Providing performance feedback to teachers: A review. *Teacher education and special education* 27, 4 (2004), 396–407.

[41] Dale H. Schunk. 1995. Self-efficacy, motivation, and performance. *Journal of Applied Sport Psychology* 7, 2 (1995), 112–137. https://doi.org/10.1080/10413209508406961 arXiv:https://doi.org/10.1080/10413209508406961

[42] Preya Shabrina, Samiha Marwan, Min Chi, Thomas W Price, and Tiffany Barnes. 2020. The Impact of Data-driven Positive Programming Feedback: When it Helps, What Happens when it Goes Wrong, and How Students Respond. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop @ EDM'20*.

[43] Valerie J Shute. 2008. Focus on formative feedback. *Review of educational research* 78, 1 (2008), 153–189.

[44] Daniel Toll, Anna Wingkvist, and Morgan Ericsson. 2020. Current State and Next Steps on Automated Hints for Students Learning to Code. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–5.

[45] Wengran Wang, Rui Zhi, Alexandra Milliken, Nicholas Lytle, and Thomas W. Price. 2020. Crescendo: Engaging Students to Self-Paced Programming Practices. In *To be published in the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. ACM, New York, NY.

[46] Philip H Winne and Allyson F Hadwin. 2013. nStudy: Tracing and supporting self-regulated learning in the Internet. In *International handbook of metacognition and learning technologies*. Springer, 293–308.