Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science

Samiha Marwan
samarwan@ncsu.edu
North Carolina State University

Ge Gao
ggao5@ncsu.edu
North Carolina State University

Susan Fisk
sfisk@kent.edu
Kent State University

Thomas W. Price
twprice@ncsu.edu
North Carolina State University

Tiffany Barnes
tmbarnes@ncsu.edu
North Carolina State University

ABSTRACT

Prior work suggests that novice programmers are greatly impacted by the feedback provided by their programming environments. While some research has examined the impact of feedback on student learning in programming, there is no work (to our knowledge) that examines the impact of adaptive immediate feedback within programming environments on students' desire to persist in computer science (CS). In this paper, we integrate an adaptive immediate feedback (AIF) system into a block-based programming environment. Our AIF system is novel because it provides personalized positive and corrective feedback to students in real time as they work. In a controlled pilot study with novice high-school programmers, we show that our AIF system significantly increased students' intentions to persist in CS, and that students using AIF had greater engagement (as measured by their lower idle time) compared to students in the control condition. Further, we found evidence that the AIF system may improve student learning, as measured by student performance in a subsequent task without AIF. In interviews, students found the system fun and helpful, and reported feeling more focused and engaged. We hope this paper spurs more research on adaptive immediate feedback and the impact of programming environments on students' intentions to persist in CS.

KEYWORDS

Programming environments, Positive feedback, Adaptive feedback, Persistence in CS, Engagement

ACM Reference Format:
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 2020 International Computing Education Research Conference (ICER '20), August 10–12, 2020, Virtual Event, New Zealand.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3372782.3406264

ICER '20, August 10–12, 2020, Virtual Event, New Zealand

ACM ISBN 978-1-4503-7092-9/20/08…$15.00
https://doi.org/10.1145/3372782.3406264

1 INTRODUCTION

Effective feedback is an essential element of student learning [16, 58] and motivation [46], especially in the domain of programming [17, 26, 40]. When programming, students primarily receive feedback from their programming environment (e.g., compiler error messages). Prior work has primarily focused on how such feedback can be used to improve students' cognitive outcomes, such as performance or learning [9, 26, 40]. However, less work has explored how such feedback can improve students' affective outcomes, such as engagement and intention to persist in computer science (CS). These outcomes are especially important because we are facing a shortage of people with computational knowledge and programming skills [19], which will not be addressed–no matter how much students learn about computing in introductory courses–unless more students choose to pursue computing education and careers.

It is also important to study feedback in programming environments because prior work shows that it can sometimes be frustrating, confusing, and difficult to interpret [9, 53, 54]. In particular, there is a need for further research on how programming feedback can be designed to create positive, motivating, and engaging programming experiences for novices, while still promoting performance and learning. Creating these positive experiences (including enjoyment and feelings of ability) are particularly important because they have a profound impact on students' intention to persist in computing [35].

In this paper, we explore the effects of a novel adaptive immediate feedback (AIF) system on novice programming students. We designed the AIF system to augment a block-based programming environment with feedback aligned with Scheeler et al's guidance that feedback should be immediate, specific, positive, and corrective [57]. Thus, our AIF provides real-time feedback adapted to each individual student's accomplishments on their performance on a specific open-ended programming task. Since our AIF system is built on data from previous student solutions to the same task, it allows students to approach problem solving in their own way. Given the beneficial impact of feedback on learning [59], we hypothesize that our AIF system will improve student performance and learning. We also hypothesize that our AIF system will improve the coding experience of novice programmers, making it more likely that they will want to persist in CS. This is especially important given the aforementioned dearth of workers with computing skills, and the

fact that many students with sufficient CS ability choose not to major in CS [31].

We performed a controlled pilot study with 25 high school students, during 2 summer camps, to investigate our primary research question: What impact does adaptive immediate feedback (AIF) have on students' perceptions, engagement, performance, learning, and intentions to persist in CS? In interviews, students found AIF features to be engaging, stating that it was fun, encouraging, and motivating. Our quantitative results show that in comparison to the control group, our AIF system increased students' intentions to persist in CS, and that students who received the AIF were significantly more engaged with the programming environment, as measured by reduced idle time during programming. Our results also suggest that the AIF system improved student performance by reducing idle time, and that the AIF system may increase novice students' learning, as measured by AIF students' performance in a future task with no AIF.

In sum, the key empirical **contributions** of this work are: (1) a novel adaptive immediate feedback system, and (2) a controlled study that suggests that programming environments with adaptive immediate feedback can increase student engagement and intention to persist in CS. Since the AIF is built based on auto-grader technologies, we believe that our results can generalize to novices learning in other programming languages and environments. This research also furthers scholarship on computing education by drawing attention to *how* programming environments can impact persistence in computing.

2 RELATED WORK

Researchers and practitioners have developed myriad forms of automated programming feedback. Compilers offer the most basic form of syntactic feedback through error messages, which can be effectively enhanced with clearer, more specific content [8, 9]. Additionally, most programming practice environments, such as CodeWorkout [23] and Cloudcoder [29], offer feedback by running a student's code through a series of test cases, which can either pass or fail. Other autograders (e.g. [5, 62]) use static analysis to offer feedback in block-based languages, which do not use compilers. Researchers have improved on this basic correctness feedback with adaptive on-demand help features, such as misconception-driven feedback [26], expert-authored immediate feedback [27], and automated hints [52, 56], which can help students identify errors and misconceptions or suggest next steps. This additional feedback can improve students' performance and learning [17, 26, 40]. However, despite these positive results, Aleven et al. note that existing automated feedback helps "only so much" [1]. In this section, we explore how programming feedback could be improved by incorporating best practices, and why this can lead to improvements in not only cognitive but also affective outcomes.

Good feedback is critically important for students' cognitive and affective outcomes [46, 47, 58] – but what makes feedback good? Our work focuses on task-level, formative feedback, given to students as they work, to help them learn and improve. In a review of formative feedback, Shute argues that effective feedback is non-evaluative, supportive, timely, and specific [58]. Similarly, in a review on effective feedback characteristics, Scheeler et al. noted that feedback should be immediate, specific, positive, and corrective to promote lasting change in learners' behaviors [57]. Both reviews emphasize that feedback, whether from an instructor or a system, should support the learner and provide timely feedback according to students' needs. However, existing programming feedback often fails to address these criteria.

Timeliness: Cognitive theory suggests that immediate feedback is beneficial, as it results in more efficient retention of information [50]. While there has been much debate over the merits of immediate versus delayed feedback, immediate feedback is often more effective on complex tasks, when students have less prior knowledge [58], making it appropriate for novice programmers. In most programming practice environments, however, students are expected to work without feedback until they can submit a mostly complete (if not correct) draft of their code. They then receive *delayed feedback*, from the compiler, autograder or test cases. Even if students choose to submit code before finishing it, test cases are not generally designed for evaluating partial solutions, as they represent correct behavior, rather than subgoals for the overall task. Other forms of feedback, such as hints, are more commonly offered *on-demand* [41, 56]. While these can be used for immediate feedback, this requires the student to recognize and act on their need for help, and repeated studies have shown that novice programmers struggle to do this [39, 53]. In contrast, many studies with effective programming feedback have used more immediate feedback. For example, Gusukuma et al. evaluated their immediate, misconception-driven feedback in a controlled study with undergraduate students, and found it improved students performance on open-ended programming problems [26].

Positiveness: Positive feedback is an important way that people learn, through confirmation that a problem-solving step has been achieved appropriately, e.g. "Good move" [7, 21, 25, 36]. Mitrovic et al. theorized that *positive feedback* is particularly helpful for novice programmers, as it reduces their uncertainty about their actions [43], and this interpretation is supported by cognitive theories as well [3, 4]. However, in programming, the feedback students receive is rarely positive. For example, enhanced compiler messages [9] improve on how *critical* information is presented, but offer no additional feedback when students' code compiles correctly. Similarly automated hints [40] and misconception-driven feedback [26] highlight what is *wrong*, rather than what is *correct* about students' code. This makes it difficult for novices to know when they *have* made progress, which can result in students deleting correct code, unsure if it is correct [45]. Two empirical studies in programming support the importance of positive feedback. Fossati et al. found that the iList tutor with positive feedback improved learning, and students liked it more than iList without it [25]. An evaluation of SQL tutor showed that positive feedback helped students master skills in less time [43].

While there is a large body of work exploring factors influencing students' affective outcomes, like retention [10] or intentions to persist in CS major [6], few studies have explored *how* automated tools can improve these outcomes [24]. There is ample evidence that immediate, positive feedback would be useful to students; however, few feedback systems offer either [25]. More importantly, evaluations of these systems have been limited only to *cognitive* outcomes;

however, prior work suggests that feedback should also impact students' affective outcomes such as their engagement and intention to persist. For example, a review of the impact of feedback on persistence finds that positive feedback "increases motivation when people infer they have greater ability to pursue the goal or associate the positive experience with increased goal value" [24]. Positive feedback has also been found to improve student confidence, and is an effective motivational strategy used by human tutors [34]. Additionally, in other domains feedback has been shown to increase students' engagement [49]. This suggests not only the need for the design of feedback that embraces these best practices, but also evaluation of its impact on cognitive and affective outcomes.

3 ADAPTIVE IMMEDIATE FEEDBACK (AIF) SYSTEM

Our adaptive immediate feedback (AIF) system was designed to provide high-quality feedback to students as they are learning to code in open-ended programming tasks (e.g. PolygonMaker and DaisyDesign, described in more detail in Section 4.2). Our AIF system continuously and adaptively confirms when students complete (or break) meaningful objectives that comprise a larger programming task. Importantly, a student can complete AIF objectives without having fully functional or complete code. This allows us to offer positive and corrective feedback that is immediate and specific. In addition, our AIF system includes pop-up messages tailored to our student population, since personalization is key in effective human tutoring dialogs [12, 21] and has been shown to improve novices' learning [33, 44].

Our AIF system consists of three main components to achieve real-time adaptive feedback: objective detectors, a progress panel, and pop-up messages. The *objective detectors* are a set of continuous autograders focused on positive feedback, that continuously check student code in real time to determine which objective students are working on and whether they have correctly achieved it or not. The *progress panel* is updated by the continuous objective detectors to color each task objective according to whether they are complete (green), not started (grey), or broken (red), since prior research suggests that students who were uncertain often delete their correct code [22]. The *pop-up messages* leverage the objective detectors to provide achievement pop-ups when objectives are completed, and motivational pop-ups when a student has not achieved any objectives within the last few minutes. These pop-ups promote confidence by praising both accomplishment and perseverance, which may increase students' persistence [15]. We strove to make the AIF system engaging and joyful, which may increase students' motivation and persistence [30].

To develop our AIF system, we first developed task-specific objective detectors, which can be thought of as continuous real-time autograders, for each programming task used in our study. Unlike common autograders which are based on instructors' test cases, our objective detectors are hand-authored to encompass a large variety of previous students' correct solutions matching various students' mindsets. To do so, two researchers with extensive experience in grading block-based programs, including one instructor, divided each task into a set of 4-5 objectives that described features of a correct solution, similar to the process used by Zhi et al. [65].

Figure 1: Adaptive Immediate Feedback (AIF) system with pop-up message (top) and progress dialog (bottom right), added to iSnap [52].

Table 1: Examples of Pop-up Messages in the AIF System.

State	message
< 1/2 objectives complete	- You are legit amazing!! 🤩
> 1/2 objectives complete	- You're on fire! 🔥🔥🤩
All objectives are done	- High FIVE!! 🙌, you DID IT!!
Fixed broken objective	- Yay!!!! 🎉it's fixed 💪😎!!
Struggle/Idle , <half done	- Yeat it till you beat it!! 👊🤓
Struggle/Idle, >half done	- You are doing great so far!! ✌️

Then, for each task, we transferred prior students' solutions into abstract syntax trees (AST). Using these ASTs, we detected different patterns that resemble a complete correct objective, and accordingly, we developed objective detectors to detect the completion for each objective. Finally, we tested and enhanced the accuracy of the objective detectors by manually verifying their performance and refining them until they correctly identified objective completion on 100 programs written by prior students.

Based on the objective detectors, we designed the progress panel to show the list of objectives with colored progress indicators, as shown in the bottom right of Figure 1. Initially, all the objectives are deactivated and grey. Then, while students are programming, the progress panel adaptively changes its objectives' colors based on students' progress detected by our objective detectors. Once an objective is completed, it becomes green, but if it is broken, it changes to red.

We then designed personalized AIF pop-up messages. We asked several high school students to collaboratively construct messages for a friend to (1) praise achievement upon objective completion, or (2) provide motivation when they are struggling or lose progress. The final messages shown in Table 1 include emojis added to increase positive affect [20, 55]. In real time, our AIF system selects a contextualized pop-up message based on students' code and actions, detected by our objective detectors. The pop-up messages provided immediate adaptive feedback, such as "Woo, one more to go!!" for a student with just one objective left, or "Good job, you FIXED it!!

;)" when a student corrected a broken objective. To praise perseverance, AIF pop-ups are also shown based on time, either after some idle time, or if a student takes longer than usual on an objective based on previous student data. It may be especially important to provide affective support to students who may be struggling. For example, if a student stopped editing for more than 2 minutes[1] and they are half-way through the task, one motivational pop-up message is "Keep up the great work 💪!!".

Our novel AIF system is the first such system to include continuous real-time autograding through objective detectors, and is the first to use such detectors to show students a progress panel for task completion in open-ended programming tasks. Further, our AIF pop-up message system is the first such system that provides both immediate, achievement-based feedback as well as adaptive encouragement for students to persist. The AIF system was added to iSnap, a block-based programming environment [52], but could be developed based on autograders for most programming environments.

3.1 Illustrative Example: Jo's Experience with the AIF System

To illustrate a student's experience with the AIF system and make its features more concrete, we describe the observed experience of Jo, a high school student who participated in our study, as described in detail in Section 4. On their first task to create a program to draw a polygon (PoygonMaker), Jo spent 11 minutes, requested help from the teacher, and received 3 motivational and 2 achievement pop-up messages. As is common with novices new to the block-based programming environment, Jo initially spent 3 minutes interacting with irrelevant blocks. Jo then received a positive pop-up message, "Yeet it till you beat it 👊🤓". Over the next few minutes, Jo added 3 correct blocks and received a few similar encouraging pop-up messages. Jo achieved the first objective, where AIF marked it as complete and showed the achievement pop-up "You are on fire! 🔥🔥." Jo was clearly engaged, achieving 2 more objectives in the next minute. Over the next 3 minutes, Jo seemed to be confused or lost, repetitively running the code with different inputs. After receiving the motivational pop-up "You're killing it! 👊😎," Jo reacted out loud by saying "that's cool". One minute later, Jo completed the 4th objective and echoed the pop-up "Your skills are outta this world 😎!!, you DID it!! 🎉,"' saying, "Yay, I did it." Jo's positive reactions, especially to the pop-up messages, and repeated re-engagement with the task, indicate that AIF helped this student stay engaged and motivated. This evidence of engagement aligns with prior work that measures students' engagement with a programming interface by collecting learners' emotions during programming [38].

Jo's next task was to draw a DaisyDesign, which is more complex, and our adaptive immediate feedback seemed to help Jo overcome difficulty and maintain focus. AIF helped Jo stay on task by providing a motivational pop-up after 3 minutes of unproductive work. In the next minute, Jo completed the first two objectives. AIF updated Jo's progress and gave an achievement pop-up, "You're the G.O.A.T[2] ✓🤩". Jo then did another edit, and AIF marked a previously-completed objective in red – demonstrating its immediate corrective feedback. Jo immediately asked for help and fixed the broken objective. After receiving the achievement pop-up "Yeet, gottem!! 👏," Jo echoed it out loud. Jo spent the next 13 minutes working on the 3rd objective with help from the teacher and peers, and 3 motivational AIF pop-ups. While working on the 4th and final objective over the next 5 minutes, Jo broke the other three objectives many times, but noticed the progress panel and restored them immediately. Finally, Jo finished the DaisyDesign task, saying "the pop-up messages are the best." This example from a real student's experience illustrates how we accomplished our goals to improve engagement (e.g. maintaining focus on important objectives), student perceptions (e.g. stating "that's cool"), performance (e.g. understanding when objectives were completed), and programming behaviors (e.g. correcting broken objectives).

4 METHODS

We conducted a controlled, pilot study during two introductory CS summer camps for high school students. Our primary **research question** is: What impact does our adaptive immediate feedback (AIF) system have on novice programmers? Specifically, we hypothesized that the AIF system would be positively perceived by students (H1-qual) and that the AIF system would increase: students' intentions to persist in CS (H2-persist), student' engagement (H3-idle), programming performance (H4-perf), and learning (H5-learning). We investigated these hypotheses using data gleaned from interviews, system logs, and surveys.

4.1 Participants

Participants were recruited from two introductory CS summer camps for high school students. This constituted an ideal population for our study, as these students had little to no prior programming experience or CS courses, allowing us to test the impact of the AIF system on students who were still learning the fundamentals of coding and who had not yet chosen a college major. Both camps took place on the same day at a research university in the United States.

We combined the camp populations for analysis, since the camps used the same curriculum for the same age range. Study procedures were identical across camps. The two camps consisted of one camp with 14 participants, (7 female, 6 male, and 1 who preferred not to specify their gender) and an all-female camp with 12 participants. Across camps, the mean age was 14, and 14 students identified as White, 7 as Black or African American, 1 as Native American or American Indian, 2 as Asian, and 1 as Other. None of the students had completed any prior, formal CS classes. Our analyses only include data from the twenty-five participants who assented–and whose parents consented–to this IRB-approved study.

4.2 Procedure

We used an experimental, controlled pre-post study design, wherein we randomly assigned 12 students to the experimental group *Exp* (who used the AIF system), and 13 to the *Control* group (who used

[1]We used a threshold of 2 minutes based on instructors' feedback on students' programming behavior.

[2]G.O.A.T., an expression suggested by teenagers, stands for Greatest of All Time.

the block-based programming environment without AIF). The pre-post measures include a survey on their attitudes towards their intentions to persist in CS and a multiple choice test to assess basic programming knowledge. The teacher was unaffiliated with this study and did not know any of the hypotheses or study details, including condition assignments for students. The teacher led an introduction to block-based programming, and explained user input, drawing, and loops. Next, all students took the pre-survey and pretest.

In the experimental phase of the study, students were asked to complete 2 consecutive programming tasks (1: PolygonMaker and 2: DaisyDesign)[3]. Task1, PolygonMaker, asks students to draw any polygon given its number of sides from the user. Task2, DaisyDesign, asks students to draw a geometric design called "Daisy" which is a sequence of overlapping n circles, where n is a user input. Both tasks required drawing shapes, using loops, and asking users to enter parameters, but the DaisyDesign task was more challenging. Each task consisted of 4 objectives, for a total of 8 objectives that a student could complete in the experimental phase of the study. Students in the experimental group completed these tasks with the AIF system, while students in the control group completed the same task without the AIF system. All students, in both the experimental and control groups, were allowed to request up to 5 hints from the iSnap system [52], and to ask for help from the teacher. In sum, there were eight objectives (as described in Section 3) that students could complete in this phase of the experiment (the experimental phase). We measured a student's programming performance based on their ability to complete these eight objectives (see Section 4.3, below for more details).

After each student reported completing both tasks[4], teachers directed students to take the post-survey and post-test. Two researchers then conducted semi-structured 3-4 minute interviews with each student. During the interviews with *Exp* students, researchers showed students each AIF feature and asked what made it more or less helpful, and whether they trusted it. In addition, the researchers asked students' opinions on the AIF design and how it could be improved.

Finally, all students were given 45 minutes to do a third, similar, but much more challenging, programming task (DrawFence) with 5 objectives *without* access to hints or AIF. Learning was measured based on a student's ability to complete these five objectives (see subsection 4.3, below for more details).

4.3 Measures

Pretest ability - Initial computing ability was measured using an adapted version of Weintrop, et al.'s commutative assessment [64] with 7 multiple-choice questions asking students to predict the outputs for several short programs. Across both conditions, the mean pre-test score was 4.44 (SD = 2.39; min = 0; max = 7).

Engagement - Engagement was measured by using the percent of programming time that students' spent idle (i.e. not engaged) on tasks 1 and 2. While surveys are often used to measure learners' engagement, these self-report measures are not always accurate, and our fine-grained programming logs give us more detailed insight into the exact time students were, and were not, engaged with programming. To calculate percent idle time, we defined idle time as a period of 3 or more minutes[5] that a student spent without making edits or interacting with the programming environment, and divided this time by the total time a student spent programming. A student's total programming time was measured from when they began programming to task completion, or the end of the programming session if it was incomplete. While we acknowledge that some "idle" time *may* have been spent productively (e.g., by discussing the assignment with the teacher or peers), we observed this very rarely (despite Jo's frequent help from friends and the teacher). Across both conditions, the mean percent idle time on tasks 1 and 2 was 13.9% (Mean = 0.139; SD = 0.188; min = 0; max = 0.625).

Programming Performance - Programming performance was measured by objective completion during the experimental phase of the study (i.e., the first eight objectives in which the experimental group used the AIF system). Each observed instance of programming performance was binary (e.g., the object was completed [value of '1'] or the object was not completed [value of '0']). This led to a repeated measures design, in which observations (i.e., whether an object was or was not completed) were nested within participants, as each participant attempted 8 objectives (in sum, 200 observations were nested within 25 participants). We explain our analytical approach in more detail below. Across both conditions, the mean programming performance was 0.870 (min = 0 ; max = 1).

Learning - Learning was measured by objective completion during the last phase (the learning phase) of the study, on the last five objectives in which neither group used the AIF system, because we assumed that students who learned more would perform better on this DrawFence task. Each observed instance of programming performance was binary. This led to a repeated measures design, in which observations (e.g., whether an objective was or was not completed) were nested within participants, as each participant attempted 5 objectives. In sum, 125 observations were nested within 25 participants. Across both conditions, the mean learning score was 0.60 (min = 0; max = 1), meaning that, on average, students completed about 3 of the 5 objectives.

Intention to Persist - CS persistence intentions were measured using the pre- and post-surveys using 7-point Likert scales adapted from a survey by Correll et al. [18]. Students were asked to state how likely they were to: 1) take a programming course in the future, 2) Minor in CS, 3) Major in CS, 4) Apply to graduate programs in CS, and 5) Apply for high-paying jobs requiring high levels of Computer Science ability. We added these measures together for a CS persistence index with a high alpha (a = 0.85)[6], indicative of the scale's reliability, with a mean of 24.8 (SD = 5.15; min = 13; max = 32).

While we could not measure actual persistence in CS, intentions to persist are a good proxy, given that research finds that they are predictive of actual persistence in STEM fields and "...hundreds of research efforts occurring [since the late 1960s] support the

[3]Programming tasks' instructions are available at: https://go.ncsu.edu/study_instructions_icer20

[4]While all students reported completing both tasks, we found some students did not finish the tasks after looking at their log data.

[5]We choose this 3-minute cutoff based on our analysis of prior student programming log data on the same tasks.

[6]Cronbach's alpha is a measure of the reliability, or internal consistency, of a scale [32].

contention that intention is the 'best' predictor of future behavior" [42].

4.4 Analytical Approach

Our quantitative analytical approach was informed by our small sample size (caused by the low number of students available for study recruitment) in two ways. First, to control for pre-existing differences between students, we control for a student's pretest performance in all of our models, as random assignment in a small sample may not be enough to ensure roughly equal levels of ability in both groups. Second, we use linear mixed effects models to maximize our statistical power when we have repeated measures (for instance, we treat each programming performance item [objective complete or incomplete] as our unit of analyses of student performance and learning). These models, "...are an extension of simple linear models to allow both fixed and random effects, and are particularly used when there is non independence in the data, such as arises from a hierarchical structure" [14]. This is appropriate to use because our data have a nested structure, as observations (e.g., CS persistence intentions, objective completion) are nested within students. Thus, observations are not independent, given that each participant contributed numerous observations. A mixed model allows us to account for the lack of independence between observations while still taking advantage of the statistical power provided by having repeated measures. It also allows us to estimate a random effect for each student, meaning that the model more effectively controls for idiosyncratic participant differences, such as differences in incoming programming ability between participants.

We use a specific type of linear mixed effects model, a linear probability model (LPM) with mixed effects[7] to predict the binary outcomes of programming performance and learning. While logistic models are typically used to predict binary outcomes, we used a LPM with mixed effects because the interpretation of coefficients is more intuitive [61]. This has led many researchers to suggest using LPMs [2], especially because they are typically as good as logistic models at predicting dichotomous variables, and their p-values are highly correlated [28].

5 RESULTS

We first use interviews to investigate (H1-qual), that the AIF system would be positively perceived by students, and then use the student surveys to determine the impact of the AIF system on intentions to persist in CS (H2-persist). Next, we analyze student log files to investigate the impact of the AIF system on student engagement (H3-idle), programming performance (H4-perf), and learning (H5-learning).

5.1 H1-qual: AIF Perceptions

To investigate student perceptions of the AIF system, we transcribed interviews of all 12 students in the *Exp* group. Afterwards, we followed a 6-step thematic analysis, described in [13, 37], to identify positive and negative themes for the AIF pop-ups and progress panel features. Two of the present authors start by (1) getting familiar with the data, then (2) generate initial codes (for each AIF feature), and then start (3) looking for dominating themes. Afterwards, one author (4) reviews the themes, and (5) refines them to focus mainly on the positive and negative aspects of each AIF feature. We then (6) combine these findings in the following summary. The main theme for each feature is whether it is helpful or not. We select quotes that represent typical positive and negative helpfulness comments from student participants labeled with s1-s12.

Pop-up messages: When asked what made pop-up messages helpful or less helpful, 10 out of 12 students agreed that pop-up messages were helpful and elaborated on why. Most students found them "engaging" [s2], "funny" [s1, s2], "encouraging" [s7], and "motivating" [s1, s5, s8]. For example, s1 stated, "it is better than just saying 'correct' because it gets you more into it," and s5 said, "it kept me focused and motivated, I was like 'yea I did it'." One student, s7, noted that messages encouraged perseverance by removing uncertainty: "especially when you don't know what you were doing, it tells that you are doing it, so it let you continue and keep going." Two students noted that pop-ups were helpful by "keeping me on track" [s3, s4]. When asked what made pop-up messages *less* helpful, two students stated that "they were not really helpful" [s4, s3], and one student s7 suggested "a toggle to [turn them] on or off". Overall, the majority of students (83%) found the pop-ups helpful and engaging.

Progress Panel: When we asked students "what about the progress panel makes it helpful or less helpful?", all students found it helpful and requested to have it in future tasks. Students said the progress panel not only helped them to keep track of their progress, but also motivated them:"it told you what you completed so it kinda gives me motivation" [s8]. In addition, students appreciated the change in colors of each objective "because you can see how much you have to do, or if you took off something [a block] and you thought it was wrong and it turns red then that means you were actually right" [s7]. When students were asked what about the progress panel makes it *less* helpful, one student incorrectly noted, "I think there was [only] one way to complete these objectives" [s6]. Students overwhelmingly found the AIF progress panel to be beneficial, especially in understanding when an objective was complete or broken, supporting Scheeler's suggestion that feedback should be specific, immediate and positive or corrective.

This thematic analysis **supports H1-qual**, that students would positively perceive the AIF system. In addition, these results provide insights on how to design personalized adaptive, immediate feedback that can engage and motivate novice high school students in their first programming experiences.

5.2 H2-persist: Intentions to Persist

To investigate H2-persist, we analyze students' survey responses to determine the impact of the AIF system on students' intentions to persist in CS. We use a linear mixed-effects model, as each student rated their intentions to persist in CS twice (once in the pre-survey and once in the post-survey). In Table 2, time takes on a value of '0' for the pre-survey and '1' for the post-survey. AIF takes on a value of '0' for all observations at time 0 (as no students had experienced the AIF system at this time), and takes on a value of '1' at time 1 if the student was in the treatment group and received the AIF system. Pretest takes on the value of the student's pretest score, in order to control for students' initial programming ability. We

[7] A LPM is simply a linear model used to predict a binary outcome.

Table 2: Estimated coefficients (Standard Error) of linear mixed models with repeated measures predicting CS persistence intentions.[8]

	Coeff. (Std. Err.)
AIF	2.234 (0.968)*
Time	-0.472 (0.682)
Pretest	0.502 (0.394)
Intercept	22.253 (1.995)***
Observations	50

Significant codes (p <): + = 0.1, * = 0.05, ** = 0.01, *** = 0.001.

find that neither time (p = 0.489) nor pretest score (p = 0.203) has a statistically significant impact on students' intentions to persist in CS, as shown in Table 2.

We also find that the AIF system significantly improves CS persistence intentions. Our linear mixed-effects model (which controls for pre-existing differences in ability between students) predicts a CS persistence score of 24.26 at time 0 (before students had completed any programming tasks) for students with an average pretest score of 4. However, at time 1 (after students had completed programming tasks), the model predicts a CS persistence score of 23.79 for students in the control condition and 26.02 for students who received the AIF system. This is a difference of 2.23 points (p = 0.021), amounting to CS persistence intentions that are about 9.37% higher for students of average ability who received the AIF system (Table 2). This provides support for H2-persist, as the AIF system improves CS persistence intentions.

5.3 H3-idle: Engagement

We use an ordinary least squares (OLS) linear regression model to investigate the impact of the AIF treatment on student engagement (H3-idle), as shown in Table 3. Engagement was measured by using idle time (discussed previously in Section 4.3), which was ascertained using programming log data. An OLS linear regression model was used instead of a t-test because it allowed us to control for pretest score. We did not use a linear *mixed* model, as there was no need to account for a lack of independence among observations because there was only one observation of engagement (idle time) per student. In Table 3, we predict the percent of total programming time a student spent idle controlling for condition and pretest score (see subsection 5.2, above, for details on coding). We find that pretest scores do not have a statistically significant impact on engagement (p = 0.533).

Importantly, we find that students who used the AIF system spent significantly less time idle (p = 0.013). Our model predicts that students with an average pretest score of 4 spent 22.6% of their programming time idle if they were in the control group, versus only 3.6% if they were in the AIF group. This means that the AIF system had a substantial impact on student engagement, as it reduced idle time by 84.2% for students with average pretest ability. In sum, we found **strong support for H3-idle**, that the AIF system improves student engagement.

[8]Within-group errors were modeled to have an autoregressive structure with a lag of 1, given the time-lag between observations.

Table 3: Estimated coefficients (Standard Error) of OLS linear regression models predicting student engagement (measured as percentage of programming time spent idle).

	Coeff. (Std. Err.)
AIF	-0.191 (0.078)*
Pretest	0.010 (0.015)
Intercept	0.188 (0.075)*
Observations	25

Significant codes (p <): + = 0.1, * = 0.05, ** = 0.01, *** = 0.001.

5.4 H4-perf: Programming Performance

We investigate H4-perf (that the AIF system improved programming performance), by analyzing programming log data on the first two tasks. We use a linear probability model with mixed effects to predict the likelihood that a student completed an objective ('1' = completed, '0' = not completed) during the experimental phase of the study (i.e., the first eight objectives in which the experimental group used the AIF system). Thus, a total of 200 observations (i.e., whether a student completed a given objective) were nested within the 25 participants for this analysis. We predict the likelihood that a student completed an objective, using the student's pre-test score and the treatment as predictors (see subsection 5.2, above, for details on coding) (Model A, Table 4). We find that a student's pre-test score has no effect on their likelihood of completing an objective (p = 0.315). However, the AIF system has a marginally statistically significant impact on programming performance (p = 0.098), as students in the *Exp* condition were 13.1 percentage points more likely to complete an objective than students in the control condition. Thus, an average student (as measured by pretest score = 4) would be expected to complete 81.4% of the objectives if they were in the control condition and 94.5% of the objectives if they were in the AIF condition. This provides marginal support for H4-perf, that the AIF system would improve students' performance.

5.5 H5-learning: Learning

To investigate H5-learning, that the AIF system improves learning, we examine student performance on the last phase of the study (i.e., on task 3, in which neither group used the AIF system to complete the last five objectives). We again use a linear mixed effects model to predict the likelihood that a student completed an objective ('1' = completed, '0' = not completed). We predict the likelihood that a student completed an objective controlling for a student's pre-test score and the treatment (see subsection 5.2, above, for details on coding) (Model B, Table 4). We again find no effect of a student's pre-test score on their likelihood of completing an objective (p = 0.952). Importantly, students receiving the AIF treatment were 25.4 percentage points more likely to complete an objective than students in the control condition, but this difference was only marginally significant (p = 0.056). Thus, a student with a pretest score of 4 would be expected to complete 47.7% of the objectives if they were in the control condition and 73.1% of the objectives if they were in the AIF condition. This provides marginal support for H5-learning, that the AIF system would improve students' learning.

Table 4: Estimated coefficients (Standard Error) of linear probability models (LPM) with mixed effects and repeated measures predicting the likelihood that a student completed an objective.

	Experimental Phase (Part 1)	Learning Phase (Part 2)
	Model A	**Model B**
Pretest	-0.017 (0.017)	0.002 (0.029)
AIF	0.131 (0.079)$^{+}$	0.254 (0.135)$^{+}$
Intercept	0.882 (0.083)***	0.470 (0.142)***
Observations	200	125

Significant codes (p <): + = 0.1, * = 0.05, ** = 0.01, *** = 0.001.

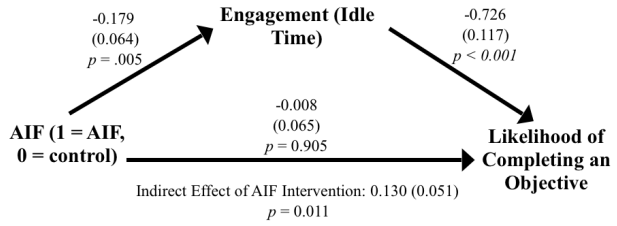

Figure 2: Mediation test results for effect of AIF intervention on idle time and likelihood of completing an objective during the experimental phase. Model controls for the effect of pretest scores. Clustered robust standard errors are shown in parentheses. N = 200 observations from 25 students.

5.6 Exploratory Mediation Analyses

We conduct an exploratory, post-hoc mediation analysis to investigate whether the AIF system may have improved objective completion in the experimental phase (part 1, with 200 observations) *because* it increased student engagement. We constructed a path model [60] to conduct our mediation analysis, allowing us to statistically suggest causal relationships in our variables: AIF, Engagement, and Objective completion. Controlling for pretest score, we found that once engagement is taken into account, the AIF system did not have a *direct* effect on objective completion (coeff. = -0.008, p = 0.905). Instead, we found evidence for an indirect effect of the AIF system on objective completion *through* the impact of the AIF system on engagement. Figure 2 illustrates the results from our path model, which finds that the AIF system reduces idle time by 17.9 percentage points (p = 0.005), and that idle time has a large, negative impact (coeff. = -0.727, p < 0.001) on the likelihood that a student completes an objective. Moreover, our mediation analysis revealed the size of the indirect effect of the AIF system: students who receive the AIF system are 13.0 percentage points more likely to complete an objective (p = 0.011) because the AIF system increased their engagement.

6 DISCUSSION

Our results provide compelling evidence that AIF can significantly improve students' intention to persist in CS. This impact of the AIF system is especially important, given that our participants had not yet entered university or declared a major, and thus the use of our AIF system could entice more students to study CS. In addition, these findings are important to the CS education community, since prior work in tutoring systems in computing *mainly* focus on the impact of feedback on students' cognitive outcomes, such as learning and performance, rather than affective outcomes which can dramatically impact student decisions. To our knowledge, this is the first evidence that automated programming feedback can improve students' desire to persist in CS, and this finding is a primary contribution of this work. We believe a primary reason for this impact on student's intentions is our feedback designed to ensure that students receive positive messages and confirmation of their success [35].

Our second compelling result is that AIF significantly improved students' engagement with our programming tasks. In particular, the system dramatically reduced idle time, from almost a quarter of students' time to less than 5%. For context, this effect on idle time is larger than that found in prior work from using a block-based instead of a text-based programming environment [51], which is often suggested as an effective way to better engage novice programmers. In addition, these findings are consistent with our qualitative interviews, which suggested that the AIF helped keep students on track, letting the students know what they had completed and what there was left to do. These results are also consistent with the "uncertainty reduction" hypothesis presented by Mitrovic et al. [43], suggesting that positive feedback helps students continue working since they are more certain about their progress in an open-ended task. Based on our combined qualitative interview analysis and quantitative log data analysis, we conclude that the AIF system helped students stay engaged. Moreover, our mediation analysis suggests that the AIF significant impacts on idle time directly helped students complete more objectives as we discuss below. We believe that these impacts are a direct consequence of well-designed adaptive immediate feedback that helped students understand their meaningful successes and mistakes.

We also find suggestive evidence that the AIF system improved students' performance and learning. While only marginally significant, the effect sizes on performance and learning were moderately large, and our ability to detect them was limited by our smaller sample size (suggesting that these effects might be significant in a larger study). It seems likely that our AIF could have impacted these outcomes, given that prior work has found that well-designed, timely feedback provided to students can improve both performance and learning in programming [39, 40]. Our AIF system also provides students with affective feedback via pop up messages, that may boost student motivation just when it is waning. Lastly, our mediation analysis suggests that the observed improvements in students' performance may have been *because* of the reduction in idle time. This is supported by research finding that engagement matters for cognitive outcomes [11]. This suggests that incorporating more positive messaging along with other forms of feedback (e.g. misconception feedback [26]) may further improve cognitive outcomes in programming. We note that feedback must be carefully designed and may be particularly beneficial if it is designed taking into account its impact on both cognitive and affective outcomes [12].

7 IMPLICATIONS AND LIMITATIONS

We believe that the findings in this study may be generalizable for novices learning to program in other programming environments and using other programming languages. This is because the core of the AIF system is its expert-authored autograders, which monitor students' progress, and there are several programming environments that have this autograding capability [5, 48, 63]. Therefore, auto-graders for other programming languages and environments can be modified to provide similar adaptive immediate feedback. For instance, in a Python programming environment with autograding, a similar AIF design could be used to detect the completion of test cases every time a student compiles their code, and experts could add explanations to each test case (similar to the expert-authored feedback in [27]), and design encouraging pop-up messages whenever a test case is passed, or fixed. We argue that experts who have written autograders should be able to implement similar adaptive, immediate feedback in other systems, using either block- or text-based programming languages, as AIF only requires the ability to autograde students' work and track their time. These systems should offer similar benefits to students, according to learning theories [59].

This study has six main limitations. First, since all students had access to hints and instructor help, we do not know if the AIF system would be equally effective in classrooms without this additional support. However, we found little difference in hint usage between conditions, and our observations reveal that few students asked for instructor help. Second, the interviews may reflect response bias or novelty effects that may have led to more positive answers, but we tried to minimize potential bias by asking for both the positive and negative aspects of AIF features. Third, having the interview about the AIF system before task 3 could have improved students' motivation in the *Exp* group to complete the third programming task. Fourth, while in our study procedure we collected a post-test from students, we decided not to report learning gains (i.e. the difference between pretest and posttest scores) since, while doing our analysis, we found 5 students who did not submit their tests (although during the camp all students claimed they completed all the tests). Fifth, there may be other reasons that AIF reduced idle time; for example, breaking down programming tasks into smaller objectives may have made the tasks less difficult for students. However, our analysis of task 3 suggests that this potential reduction in difficulty did not hinder student learning in a subsequent task without AIF. Sixth and finally, our study had a small sample size, and it lasted for a short period of time. However, we argue that this study shows the promising potential of the impact of the AIF system, with results that are consistent with learning theories, and in our future work we plan to conduct a larger classroom study to verify the impact of AIF in different settings.

8 CONCLUSION

The **contributions** of this work are the design and development of a new adaptive immediate feedback system based on autograders, and a controlled study demonstrating that our AIF system: 1) is well-received by students, 2) improves students intention to persist in CS, 3) increases students' engagement, 4) improves students' performance, and 5) learning. Our interview results confirmed our hypothesis (H1-qual) that the AIF system would be positively perceived by students, and our survey results confirmed our hypothesis (H2-persist) that AIF would improve students' intention to persist in CS. Additionally, by investigating students' log data, we confirmed our hypothesis (H3-idle) that the AIF system would improve students' engagement during programming (as measured by students' idle time spent while programming). Moreover, from analyzing students' programming performance, our results partially support our hypotheses (H4-perf) that the AIF system would improve students' performance and (H5-learning) learning. In future work, we plan to generalize our approach to develop adaptive immediate feedback for more assignments and other programming languages. In addition, we plan to conduct larger classroom studies to investigate the impact of adaptive immediate feedback within the context of graded assignments.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grant 1623470.

REFERENCES

[1] Vincent Aleven, Ido Roll, Bruce M. McLaren, and Kenneth R. Koedinger. 2016. Help Helps, But Only So Much: Research on Help Seeking with Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education* 26, 1 (2016), 1–19.

[2] Joshua D Angrist and Jörn-Steffen Pischke. 2008. *Mostly harmless econometrics: An empiricist's companion.* Princeton university press.

[3] Susan J Ashford. 1986. Feedback-seeking in individual adaptation: A resource perspective. *Academy of Management journal* 29, 3 (1986), 465–487.

[4] Susan J Ashford, Ruth Blatt, and Don VandeWalle. 2003. Reflections on the looking glass: A review of research on feedback-seeking behavior in organizations. *Journal of management* 29, 6 (2003), 773–799.

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

[6] Lecia J Barker, Charlie McDowell, and Kimberly Kalahar. 2009. Exploring factors that influence computer science introductory course students to persist in the major. *ACM Sigcse Bulletin* 41, 1 (2009), 153–157.

[7] Devon Barrow, Antonija Mitrovic, Stellan Ohlsson, and Michael Grimley. 2008. Assessing the impact of positive feedback in constraint-based tutors. In *International Conference on Intelligent Tutoring Systems*. Springer, 250–259.

[8] Brett A Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3 (2016), 148–175.

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

[10] Maureen Biggers, Anne Brauer, and Tuba Yilmaz. 2008. Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers. *ACM SIGCSE Bulletin* 40, 1 (2008), 402–406.

[11] Phyllis C Blumenfeld, Toni M Kempler, and Joseph S Krajcik. 2006. *Motivation and cognitive engagement in learning environments.* na.

[12] Kristy Elizabeth Boyer, Robert Phillips, Michael D Wallis, Mladen A Vouk, and James C Lester. 2008. Learner characteristics and feedback in tutorial dialogue. In *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, 53–61.

[13] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[14] J. Bruin. 2011 (accessed April 6, 2020). *INTRODUCTION TO LINEAR MIXED MODELS.* https://stats.idre.ucla.edu/stata/ado/analysis/

[15] Erin Cech, Brian Rubineau, Susan Silbey, and Caroll Seron. 2011. Professional role confidence and gendered persistence in engineering. *American Sociological Review* 76, 5 (2011), 641–666.

[16] AT Corbett and John R Anderson. 1989. Feedback timing and student control in the LISP Intelligent Tutoring System. In *Proceedings of the Fourth International Conference on AI and Education*. 64–72.

[17] Albert Corbett and John R. Anderson. 2001. Locus of Feedback Control in Computer-Based Tutoring: Impact on Learning Rate, Achievement and Attitudes. In *Proceedings of the SIGCHI Conference on Human Computer Interaction*. 245–252.

[18] Shelley J Correll. 2004. Constraints into preferences: Gender, status, and emerging career aspirations. *American sociological review* 69, 1 (2004), 93–113.

[19] Peter J Denning and Edward E Gordon. 2015. A technician shortage. *Commun. ACM* 58, 3 (2015), 28–30.

[20] Daantje Derks, Arjan ER Bos, and Jasper Von Grumbkow. 2008. Emoticons and online message interpretation. *Social Science Computer Review* 26, 3 (2008), 379–388.

[21] 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.

[22] 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.

[23] Stephen H Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: short programming exercises with built-in data collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. 188–193.

[24] Ayelet Fishbach and Stacey R Finkelstein. 2012. How feedback influences persistence, disengagement, and change in goal pursuit. *Goal-directed behavior* (2012), 203–230.

[25] 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.

[26] 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.

[27] 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.

[28] Ottar Hellevik. 2009. Linear versus logistic regression when the dependent variable is a dichotomy. *Quality & Quantity* 43, 1 (2009), 59–74.

[29] David Hovemeyer and Jaime Spacco. 2013. CloudCoder: a web-based programming exercise system. *Journal of Computing Sciences in Colleges* 28, 3 (2013), 30–30.

[30] Tony Jenkins. 2001. The motivation of students of programming. In *Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2001, Canterbury, UK, June 25-27, 2001*. 53–56.

[31] Sandra Katz, David Allbritton, John Aronis, Christine Wilson, and Mary Lou Soffa. 2006. Gender, achievement, and persistence in an undergraduate computer science program. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems* 37, 4 (2006), 42–57.

[32] Rex B Kline. 2015. *Principles and practice of structural equation modeling*. Guilford publications.

[33] Michael J Lee and Andrew J Ko. 2011. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*. ACM, 109–116.

[34] Mark R Lepper, Maria Woolverton, Donna L Mumme, and J Gurtner. 1993. Motivational techniques of expert human tutors: Lessons for the design of computer-based tutors. *Computers as cognitive tools* 1993 (1993), 75–105.

[35] Colleen M Lewis, Ken Yasuhara, and Ruth E Anderson. 2011. Deciding to major in computer science: a grounded theory of students' self-assessment of ability. In *Proceedings of the seventh international workshop on Computing education research*. 3–10.

[36] R Luckin et al. 2007. Beyond the code-and-count analysis of tutoring dialogues. *Artificial intelligence in education: Building technology rich learning contexts that work, R. Luckin, KR Koedinger, and J. Greer, Eds. IOS Press* (2007), 349–356.

[37] Moira Maguire and Brid Delahunt. 2017. Doing a thematic analysis: A practical, step-by-step guide for learning and teaching scholars. *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education* 9, 3 (2017).

[38] Chris Martin, Janet Hughes, and John Richards. 2017. Designing engaging learning experiences in programming. In *International Conference on Computer Supported Education*. Springer, 221–245.

[39] 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.

[40] 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.

[41] 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.

[42] David M Merolla, Richard T Serpe, Sheldon Stryker, and P Wesley Schultz. 2012. Structural precursors to identity processes: The role of proximate social structures. *Social Psychology Quarterly* 75, 2 (2012), 149–172.

[43] 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.

[44] Roxana Moreno and Richard E Mayer. 2004. Personalized messages that promote science learning in virtual environments. *Journal of educational Psychology* 96, 1 (2004), 165.

[45] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: the good, the bad, and the quirky–a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin* 40, 1 (2008), 163–167.

[46] Susanne Narciss and Katja Huth. 2004. How to design informative tutoring feedback for multimedia learning. *Instructional design for multimedia learning* 181195 (2004).

[47] David J Nicol and Debra Macfarlane-Dick. 2006. Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in higher education* 31, 2 (2006), 199–218.

[48] Pete Nordquist. 2007. Providing accurate and timely feedback by automatically grading student programming labs. *Journal of Computing Sciences in Colleges* 23, 2 (2007), 16–23.

[49] Helen J Parkin, Stuart Hepplestone, Graham Holden, Brian Irwin, and Louise Thorpe. 2012. A role for technology in enhancing students' engagement with feedback. *Assessment & Evaluation in Higher Education* 37, 8 (2012), 963–973.

[50] Gary D Phye and Thomas Andre. 1989. Delayed retention effect: attention, perseveration, or both? *Contemporary Educational Psychology* 14, 2 (1989), 173–185.

[51] Thomas W. Price and Tiffany Barnes. 2015. Comparing Textual and Block Interfaces in a Novice Programming Environment. In *Proceedings of the International Computing Education Research Conference*.

[52] 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*.

[53] 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*.

[54] 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*.

[55] Monica A Riordan. 2017. Emojis as tools for emotion work: Communicating affect in text messages. *Journal of Language and Social Psychology* 36, 5 (2017), 549–567.

[56] Kelly Rivers and Kenneth R. Koedinger. 2017. Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 37–64.

[57] 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.

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

[59] Marieke Thurlings, Marjan Vermeulen, Theo Bastiaens, and Sjef Stijnen. 2013. Understanding feedback: A learning theory perspective. *Educational Research Review* 9 (2013), 1–15.

[60] Jodie B Ullman and Peter M Bentler. 2003. Structural equation modeling. *Handbook of psychology* (2003), 607–634.

[61] Paul Von Hippel. 2015. Linear vs. logistic probability models: Which is better, and when. *Statistical Horizons* (2015).

[62] Wengran Wang, Rui Zhi, Alexandra Milliken, Nicholas Lytle, and Thomas W. Price. 2020. Crescendo : Engaging Students to Self-Paced Programming Practices. In *Proceedings of the ACM Technical Symposium on Computer Science Education*.

[63] 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)*.

[64] David Weintrop and Uri Wilensky. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs.. In *ICER*, Vol. 15. 101–110.

[65] Rui Zhi, Thomas W. Price, Nicholas Lytle, Yihuan Dong, and Tiffany Barnes. 2018. Reducing the State Space of Programming Problems through Data-Driven Feature Detection. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop@ EDM*.