Unproductive Help-seeking in Programming: What it is and How to Address it

Samiha Marwan
samarwan@ncsu.edu
North Carolina State University

Anay Dombe
anay.dombe15@vit.edu
Vishwakarma Institute of Technology

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

ABSTRACT

While programming, novices often lack the ability to effectively seek help, such as when to ask for a hint or feedback. Students may avoid help when they need it, or abuse help to avoid putting in effort, and both behaviors can impede learning. In this paper we present two main contributions. First, we investigated log data from students working in a programming environment that offers automated hints, and we propose a taxonomy of unproductive help-seeking behaviors in programming. Second, we used these findings to design a novel user interface for hints that subtly encourages students to seek help with the right frequency, estimated with a data-driven algorithm. We conducted a pilot study to evaluate our data-driven (DD) hint display, compared to a traditional interface, where students request hints on-demand as desired. We found students with the DD display were less than half as likely to engage in unproductive help-seeking, and we found suggestive evidence that this may improve their learning.

ACM Reference Format:
Samiha Marwan, Anay Dombe, and Thomas W. Price. 2020. Unproductive Help-seeking in Programming: What it is and How to Address it. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20), June 15–19, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

Providing students with feedback serves as an essential element of learning theories [32, 41]. Traditionally, feedback comes from an instructor, but with the growing enrollment in Computer Science (CS) classes [5], researchers put extensive effort in developing programming environments to provide automated feedback to students for the same purpose. Several studies showed that this feedback, like simple test-case feedback [7, 18], or hints suggesting a single edit to proceed [15, 39] can increase students' performance and learning [11, 17, 22]. However, this positive impact requires the student to use help effectively, which is a difficult metacognitive skill [2]. Despite the prevalence of automated help in programming classrooms [11, 24, 39], little prior work has investigated how to encourage students to seek and use this help more effectively.

A large body of work in mathematics tutoring systems suggests that students' unproductive use of help features is a serious challenge [3, 4, 36]. For example, Aleven et al. showed that 72% of students' actions in a Geometry tutoring system represented unproductive help-seeking behavior, such as avoiding help when they needed it or abusing help to speed through the assignment [4]. Further, unproductive help-seeking predicts poor learning outcomes [2, 6], suggesting that students need additional guidance on how and when to use help. A naive approach might be to limit the amount of help provided (e.g. give no more than 3 hints). However, previous studies show that students with different prior knowledge, need different amounts of help: " the lower the prior knowledge, the higher the need for assistance" [19, 45]. Therefore, not only is it important to encourage students to effectively seek help, but it is also important to tailor the amount of help to their performance [37]. In programming environments, to design such guidance, we first need to understand how students engage in help-seeking in computing classrooms, which is unclear from prior work. To our knowledge, prior work has been limited to analyzing survey data [42], or characterizing students' motivations for seeking and avoiding help [34]. More work in the programming domain is needed to identify the specific *ways* that students unproductively seek help, to guide instructors and inform the design of feedback interfaces to address these behaviors.

In this work, we investigate the following research questions: RQ1) What specific forms of unproductive help-seeking do novices engage in when programming? RQ2) How can the design of a help interface improve students' help-seeking behavior? To address RQ1, we investigated log data, capturing real students' use of hints during independent programming homework and we propose a preliminary taxonomy of novices' unproductive help seeking behavior. Similar to Aleven et al. [2], we found the majority of students engaged in some form of unproductive help-seeking, avoiding or abusing it. To address RQ2, we designed a novel system for displaying adaptive hints. It uses a data-driven (DD) algorithm to encourage students to ask for help, based on their progress compared to prior students. We conducted a small pilot study to compare students' help-seeking behavior when working in a programming environment with our DD display, compared to a traditional on-demand hints display (i.e. hints provided upon students' request). We found students in the DD display condition never abused help and were less likely to unproductively avoid it, compared to students in the on-demand condition. We also found suggestive evidence that this improved behavior may translate into learning, but further work is needed to verify this. In addition, despite the DD display limiting students' ability to use help, students perceived it to be at least as helpful as the on-demand display, and they noted its intelligence in providing hints only when they need it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ITiCSE '20, June 15–19, 2020, Trondheim, Norway
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6874-2/20/06…$15.00
https://doi.org/10.1145/3341525.3387394

2 RELATED WORK

Prior work suggests that automated feedback can be an effective means of improving students' performance and learning when used appropriately. For example, Corbett et al. found students who received automated feedback in their ACT programming tutor performed better on a post-test than those without feedback [11]. Similarly, Marwan et al. found that students with access to adaptive hints performed better on subsequent tasks than a control group, though *only* when students were asked to self-explain the hints [22]. However, Aleven et al., noted that students generally lack the ability to seek help effectively, which obstructs the ability of learning environments with feedback to improve students' learning [2]. Therefore, in this section we review models of how students seek help, different ways that learning environments display help, and how this impacts students' help-seeking and learning.

Help-seeking is defined as a meta-cognitive skill represented in the ability of students to solicit help from a teacher, peer, or other sources [2]. Meta-cognitive skills are essential for learning [12], and have been strongly linked to students' performance in CS [9, 10]. Many help-seeking models are based on Nelson-Le Gall's help-seeking theory that states steps needed for a student to seek only the amount of help needed to complete a task, known as instrumental help-seeking [29, 30]. Reflecting Nelson-Le Gall's and Newman's help-seeking theories [29, 31], Aleven et al., presented the first help seeking model (HSM) for a computer-based learning environment (Geometry Cognitive Tutor) [2]. However, the HSM depends on estimates of students' prior knowledge which is hard to assess for *novices* in programming. Also, it is unclear how to apply HSM in programming as the model features are specific to their Geometry tutor features. In programming, Vaessen's et al., developed a Discrete Markov Model (DMM) to detect students' help-seeking strategies in a functional programming tutor. Their model predicted these strategies based on students' self-reported achievement goals, which make their model context-specific [42]. In a qualitative study, Price et al. identified *factors* affecting students' help seeking in programming problems, providing a baseline to consider when designing models for help-seeking [34]. Based on prior work, this paper took a first step to define *what is unproductive help-seeking* from students' step-by-step interactions during programming in an authentic classroom setting, which can be generalized to different programming environments.

Several classroom studies, adopting Nelson-Le Gall and Newman help-seeking theories [29, 31], found that students who need help are the least likely to ask for it [19, 35, 45]. In computer-based environments, these results motivate researchers to explore the effect of different help displays on students' help-seeking and learning. Most learning environments provide on-demand help, i.e. providing help upon students' request, like Andes (physics tutor) [43], Assisstments [36], and ITAP [39]. Prior work shows that on-demand help can improve students' learning [2, 38] because it is better to give students control over requesting help [36]. However, they do not prevent students' help abuse. For example, students may keep requesting hints, i.e. gaming the system, to finish tasks faster, without understanding the reason behind the hint. In an algebra tutor, Baker et al. found that help abusers learn only 2/3 as much as other students [6]. To address help abuse, Murray et al., studied delaying help when students request it, as cognitive tutors suggest that delaying help reduces help requests and improves students' learning [26]. Murray et al. found that delaying help eliminated help abuse; but does not affect learning. They also found the number of hints received was negatively correlated with students' post-test scores, similar to other studies in different domains [22, 25].

In addition to abusing on-demand help, avoiding help has been shown as a second serious unproductive help-seeking behavior that negatively correlates with learning [2, 3]. For example, Baker et al., using students' log data in a Genetics Cognitive Tutor, found that help avoidance is negatively associated with students' performance on a transfer test [13]. To address help avoidance, researchers allowed tutors to provide help proactively, e.g. automatically after a student makes a number of errors. Several studies, in the math domain, compared the effect of on-demand help versus proactive help on students' learning. Murray et al. found that proactive help can be more effective for some students because it can save time when a student is floundering, and provides valuable information at a time when the student is motivated to learn it [27]. In the CS domain, Corbett et al., found that students who received immediate feedback (similar to proactive feedback) in the ACT programming tutor finished exercises faster than those who received on-demand one, particularly in harder exercises [11]. Conversely, Razzaq et al. found that students learned significantly more from on-demand hints over proactive ones in a math tutor [36].

Despite the amount of work done to evaluate students' help-seeking behavior in several domains, far less work has been done in programming [11]. Research in programming tutoring environments mainly focuses on how to develop automated feedback; rather than how to display it and how students ask for it. Perhaps this is the reason why there are only a few studies that found an impact of programming feedback on students' learning [15, 22]. For example, Rivers in her Ph.D. thesis on ITAP, a Python tutor, found that the percentage of students requesting hints ranged from 26.9% - 53%, and therefore the impact of ITAP's help might not be clear due to the large number of students who avoided using help [39]. In this paper, we developed a novel data-driven display for hints combining advantages of both proactive and on-demand help display to improve students' help-seeking behavior, and it can also be deployed in different programming environments as described in Section 4.4.

3 STUDY 1: DEFINING UNPRODUCTIVE HELP-SEEKING IN PROGRAMMING

The goal of this study is to address our first research question: What specific forms of unproductive help-seeking do novices engage in when programming? From this study, we propose a preliminary taxonomy of unproductive-help seeking in programming, which informs the design of a new hint display, presented in Section 4.1.

3.1 Method

To explore *how* students in authentic classroom settings unproductively seek help, we looked at how all students (n = 50) used hints during two homework assignments in an introductory computing course for non-majors in a public university in the United States[1].

[1]We did not have access to students' demographic information.

In the first part of this course, students were studying block-based programming by solving exercises from the BJC curriculum [16]. The first homework is called "Squiral", and the other one is called "Guessing Game 2". In Squiral, students write a procedure that takes three parameters (R, L, and T) to draw a square-like spiral where the spiral has *R* rotations, of side length *L* and side thickness *T*. In Guessing Game 2, the program should generate a random secret number within a provided range and ask the user to guess this secret number, keeping track of how many guesses the user has made. The two exercises together required the use of loops, variables, conditionals, procedures and lists.

Students programmed their homeworks in iSnap [33], which is an extension to the block-based Snap*!* programming environment that provides on-demand hints. These hints suggest an edit a student can make to bring their code closer to the correct solution. iSnap also logs all students' interactions during programming (e.g. creating, or deleting a block) as a *trace*, allowing researchers to replay all of a student's steps in this trace. Students produced an average of 351 and 740 edits, on Squiral and Guessing Game 2, respectively. To capture students' authentic help-seeking behavior, we focused on homework assignments, where students work *independently* and typically use automated help, rather than instructor help. By studying students' log data, we can observe their behavior without influencing it (an instructor's presence might deter some types of help abuse). While log data has some limitations, its analysis has led to substantial research insights in computing education [2, 14, 39].

We applied a mixed-methods approach to identify and categorize types of unproductive help-seeking in our log data. Two researchers, one of whom had extensive experience analyzing students' log data in iSnap, started off by reviewing prior work on taxonomizing help-seeking behavior [2]. Afterwards, the two researchers analyzed students' log data through a 3-phase method, adapted from Dong et al. [14]: (1) Phase 1: *Initial Definition*: the two researchers independently, manually inspected half of the Squiral log data, initially focusing on students' hint requests. They noted the time taken to view each hint, whether the student followed the suggestion in the hint, and the relationship between hint requests. The researchers used this data to identify initial categories of unproductive help-seeking, and discussed these categories to produce initial definitions. (2) Phase 2: *Consensus*: the two researchers divided the second half of the first homework into two sets and each tagged students' data with the initial definitions. Finally, both researchers resolved their disagreements and discussed their findings with two other Snap*!* instructors who have previous research experience with help-seeking behavior, and then they refined their definitions accordingly. (3) Phase 3: *Verification*: the two researchers divided the second homework into two sets, and each tagged a set by the resulted categories they defined in the previous phase. Neither researcher identified any new behaviors beyond those defined in Phase 2.

3.2 Results and Discussion

We organized our results into a preliminary taxonomy of novices' unproductive help seeking in programming. As in prior work, we found that help abuse and help avoidance are the two main forms of unproductive help-seeking behavior; however, we further refined these into subcategories, which reflect distinct, but not mutually exclusive, cases of these behaviors. We compare our results to Aleven et al.'s Help Seeking Model (HSM), discussed in Section 2.

Help abuse: the behavior of asking for too much unnecessary help. Based on our data we categorized it into three subcategories:

1- Immediate Help: Students who exhibit this behavior request hints *before* beginning to program a task or objective on their own. For example, some students started their homework by clicking on the help button, and this often predicted high reliance on hints throughout the assignment. This suggests students are deliberately "gaming the system" [6], which will likely obstruct learning. When investigating help seeking in the Geometry tutor, Aleven et al. argue that an *initial* help request can be legitimate for students who have no idea where to start. However, Ko et al. suggest that for novice *programmers*, these "design barriers" are best addressed with more comprehensive help, such as examples, which is rarely provided by automated help [21]. Therefore, we argue that attempting to use low-level feedback in this case constitutes help abuse.

2- High Frequency: In this behavior, students request and consistently follow a high number of hints in a very short time (e.g. five hints in one minute). While this may accompany Immediate hint requests, it can happen at any time during the assignment.

High Frequency behavior is very similar to the help abuse form "clicking through hints" in the HSM, where a student moves to the next hint before spending enough time with the current one [1]. However, in novice programming "clicking through hints" might not always be hint abuse. Because programming problems have vast solution spaces [39], there are many relevant hints a system could provide, and novices may need to view many of them before finding a relevant one [23]. Therefore, our definition and tagging distinguished between students who viewed many hints and *followed them*, versus those who simply viewed many hints and followed one.

3- Unneeded Help: In this behavior, students ask for repeated hints, despite having demonstrated an ability to progress independently. For example, some students completed half of the solution successfully with little-to-no hint usage, and then switched to relying almost exclusively on hints, sometimes even destroying all their original code. Unlike the first two categories of abuse, whether a hint request constitutes Unneeded Help depends on the student and their current progress.

This behavior is similar to the help abuse form "Ask hint when skilled enough to try a step" in the HSM; however, the HSM defined it based on estimating students' prior knowledge [2], which is hard to measure for novices, especially in programming. We assumed students are skilled if they were making adequate progress but then decided to rely on hints without trying.

Help avoidance: this is when students fail to ask for hints when struggling and a hint could help them move forward. While our trace data allowed us to identify instances of help avoidance in an authentic homework setting, it did not give us clear insight into students' motivations for avoiding help. Therefore, we discuss two broader categories of help avoidance and discuss them in the light of prior work that investigated *factors* affecting students' help-seeking in a laboratory study [34].

1- No Use: Some students spent a long time with unproductive work with no help requests. For example, one student programmed Squiral iteratively, i.e. as a series of "move" and "turns" without

Table 1: Percent of students tagged with each help-seeking category from the Taxonomy in Study 1: HW1 and HW2, and Study 2: data-driven (DD) vs. on-demand (OD) display.

		HW1	HW2	DD	OD
Abuse	Immediate help	8%	15.21%	0%	0%
	High-freq.	20%	21.73%	0%	0%
	Unneeded help	24%	32.60%	0%	20%
Avoidance	Stopped Use	16%	4.34%	26.66%	13.33%
	No Use	14%	23.91%	0%	33.33%
	Productive	42%	26.08%	73.33%	33.33%

using any variables or loops, making a total of 40 blocks. This solution is inefficient and does not match the grading rubric since the student is not using parameters. However, the student did not request a single hint. Prior work suggested that this behavior might be because students wanted to be independent, or they were not aware of the help button[34], which suggests the need for a hint display interface that reminds students to ask for a hint.

2- Stopped Use: In a given homework, some students requested hints at the beginning, and then stopped doing so, even when they struggled. Some students even submitted their code incorrectly without asking for help again. Prior work shows that some students may have found their initial hints to be uninterpretable [23] or untrustworthy [34], and therefore avoided requesting more help.

Because this taxonomy reflects specific unproductive help-seeking behavior inferred from students' log data, it makes it applicable to design programming environments' interfaces that can deter this behavior. While we describe these behaviors in terms of hints in our data, we argue they generalize to other forms of help (e.g. enhanced compiler messages [8], and misconception feedback [17]). Table 1 shows the percentages of each unproductive help-seeking category detected from students' data. From Table 1, we found 58% and 74% of students' help-seeking behavior is unproductive, in Squiral (HW 1) and Guessing game 2 (HW 2), respectively. These alarming results motivated us to design an interface for displaying hints to moderate students' help-seeking based on our taxonomy. While this taxonomy reflects the same broad categories of help abuse and avoidance in prior work [2], it also defines more specific subcategories, where *each* inform classroom instruction and the design of help interfaces. For example, while High Frequency abuse can be addressed by limiting the frequency of hint requests, this will not address many forms of Unneeded Help.

Our results have two important limitations. First, they are based on only two assignments in a single environment with hints, though our findings' close alignment with prior work suggests they will likely generalize to other classrooms and forms of help. Second, while our use of log data captured authentic help use, it limited our ability to infer students' motivations, especially for help avoidance. However, our results represent a novel contribution to the computing education literature, as our taxonomy is the first to define unproductive help seeking behaviors, grounded in real data, focusing explicitly on programming classrooms. Also, our results can directly inform the design of novel help systems, as discussed in Section 4.1.

4 STUDY 2: IMPROVING HELP-SEEKING

This study addressed our second research question: How can the design of a help interface improve students' help-seeking behavior? To provide a solution to this question, we designed a novel Data-Driven (DD) display for hints (described in Section 4.1) to address the unproductive help-seeking behavior defined in Study 1. In addition, we conducted a pilot study to evaluate the impact of our display on students' a) help-seeking behavior, b) learning, and c) perceptions of the helpfulness of the programming environment.

4.1 Design of Data-Driven (DD) Display

We designed a novel hint display in iSnap to address each category in the two high level problems we identified, help abuse and help avoidance. A primary finding of Study 1 is that both help abuse and help avoidance can be defined – and addressed – with respect to the student's progress, rather than simply based on the frequency of help requests. Higher levels of help may be appropriate for some students and not others (e.g. "the lower the prior knowledge, the higher the need for assistance" [19, 40]). Thus, a key feature of our hint display is a way to measure a students' progress over time, and whether it represents a need for help. We do so using a data-driven (DD) approach, that measures the average time taken by prior students to complete each objective (e.g. "draw a shape") of the same task. We then measure a students' progress in real time and compare it to the average of prior students – if they have taken longer than expected to complete a given objective, we assume they may need help. Our DD approach does not rely on expert judgements of how long each objective should take, which can be biased by the expert blindspot [28]. We use an autograder, as is common in many programming environments [7, 18, 44], to assess objective completion. We use this DD progress measure to adapt when and how the hint display offers help, addressing both help abuse and avoidance.

For help-abuse, our primary hypothesis is that we can address all its three categories using our DD approach by restricting students to request hints only when they are taking longer than expected on a given objective. This addresses "Immediate Help," and partially addresses "Unneeded Help," since students cannot ask for hints until they have attempted an objective for some time. To address "High Frequency" hints abuse, we also added a 2-minute "cooldown" on hints, deriving the following rule: *At any time that a student is taking longer than expected on their current objective (based on the DD mechanism), they accumulate one hint every two minutes, which they can request whenever they want.* For example, if the system detects student Alice is not progressing on Objective 1, then she will be able to ask for a hint. If she still does not progress, she will get another hint two minutes later. If she does progress, then she won't get another one until the system detects she has fallen behind on Objective 2. Note that we chose the 2-minute threshold based on our review of log data in Study 1, but we make no claim that is generalizable. It is preferable to be adjusted based on the instructor's choice.

For help avoidance, to address the first category "No Use", we wanted to make the hints more salient, and remind students to use them without taking away students' sense of control [36]. To do so, whenever a student accumulates a hint, meaning they have

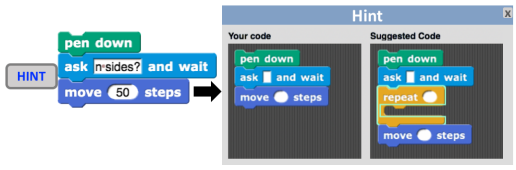

Figure 1: Example of a next-step hint given by iSnap. When the HINT button (left) is clicked, a hint is shown (right).

fallen behind and may need help, the system pops up a *flashing* hint button (as shown in Figure 1), embedded in students' code, as a subtle reminder. The student can still choose when and whether to click on the button and view the hint, combining the advantages of proactive and on-demand hints displays [27, 36]. For the second help avoidance category "Stopped Use", we did not focus in this work on improving hint quality. However, one possible way to address this behavior is to provide hints only when students need them, using the DD mechanism. As Murray et al. suggested, providing hints proactively (in our case it is just proactive reminders), may give students information at times they really need it [27], perhaps motivating them to open it.

4.2 Method

Population: Because it is hard to assign students into different learning conditions in a real classroom (i.e. issues of fairness, different students seeing different interfaces), we recruited undergraduate students from an introductory engineering course at the same university for our study. As in our classroom population, these students had not taken any prior programming course and therefore, we assumed they had minimal experience. We also allowed students to take the study independently, online, as in a homework setting, through a web interface that led them through each procedure step. We compensated participants with a $10 gift card to encourage broader participation. Our population included 30 students (16 males; 14 females). All students were 18-20 years old and in their first year. Their intended majors were Engineering (n = 21), Biology (n = 5), CS (n = 2), and undecided (n = 2).

Programming Tasks: In this study, students were asked to solve two tasks, the first one with the hints and the second without the hints. Because we wanted to explore help-seeking under varying levels of difficulty, we designed three different tasks (A, B and C), each using similar programming concepts but having increasing difficulty. We assigned half of the students to take task A, then task B, and the other half took task B, then task C. Task A asked students to take an input number n and draw a polygon with n sides. In Task B, students were asked to take an input n and draw a strip of n triangles. Task C asked students to draw a series of n "daisy squares," a geometric design made of overlapping squares. Our goal with varying the difficulty was to create scenarios where students would need more or less help, allowing us to test the adaptive levels of help provided by the DD display. However, we did not control for tasks' difficulty in our analysis. We used data from a prior experiment using tasks A and B to train the timings for the DD display.

Procedure: Students first took a short presurvey and read through a tutorial about the block-based programming environment (iSnap) that focused on the programming concepts needed for the study tasks (e.g. loops, and drawing), using both text and short animations. We randomly assigned half of the students (n = 15) to have access to on-demand hints and the other half to use our new DD hint display (n = 15)[2]. Based on student's condition, the tutorial described how iSnap can offer help to them when needed. Students were then given up to 15 minutes to complete programming Task 1, and iSnap provided them with hints based on their condition. Afterwards, they had 15 minutes to complete Task 2, this time without hints, which we use as a measure of learning transfer. After each programming task, students took a post-task survey, where they were asked some questions on their experience and the hints provided (on Task 1).

4.3 Results and Discussion

RQ2a: *What is the impact of the DD display on students' help-seeking behavior?* To understand whether the DD display reduced unproductive help-seeking, two researchers, who were blind to students' conditions, tagged each student with any form of help abuse or avoidance defined in the Taxonomy in Section 3.2. We report the full results in Table 1. In the on-demand condition, researchers tagged 7 (46.66%) as help avoiders, and 3 (20%) as help abusers, while in the DD display condition, researchers tagged 4 (26.66%) as help avoiders and 0% as help abusers. This suggests that the DD display cuts incidence of help avoidance in almost half and prevents help abuse altogether. For comparison, we combined these behaviors into a single attribute indicating unproductive (1) or productive (0) help use. A Fisher's exact test shows that the difference in unproductive help use across the two conditions is not significant ($p = 0.14$)[3] but had a moderate effect size (odds ratio = 3.92). The effect of our DD display may have been reduced by the tasks being too easy to necessitate much hint use: we found that 46.6% of students who asked for little or no hint still finished Task 1 successfully. The effect size suggests the DD display had a meaningful impact on students' help-seeking behavior, but our small sample size makes it unclear how well these results would generalize.

We then investigated how well our DD display encouraged students to ask for an appropriate number of hints, as defined by *its own algorithm*. We compared the number of hints suggested by the algorithm and the number of hints opened by students (for the on-demand condition, we simulated the DD algorithm on students' data, since they did not have the DD display). We found a much stronger correlation between the number of hints recommended by the algorithm and the number of hints opened by students in the DD condition ($r = 0.79$), than that in the on-demand condition ($r = 0.31$). This confirms that students' natural help-seeking behavior (in the on-demand condition) does not align with the algorithm's estimate of their help needs, but that the DD design succeeded in shaping students' help use. However, some students in the DD condition did still avoid help (27%). Manual investigation shows that these students were offered hints but chose not to click the hint button, sometimes after seeing a potentially confusing hint.

[2]We assigned the same number of students in each condition to each pair of tasks and so the tasks difficulty was equivalent for both conditions.
[3]We report statistical tests, along with effect sizes, to help the reader better interpret our results. However, as this was pilot study with a small sample, these tests are only likely to detect large effects and should be interpreted cautiously.

RQ2b: *How does the DD display impact students' learning?* We chose to measure students' performance on each task using the total time taken to complete the task, since 50% of students in both conditions successfully finished both tasks. If a student did not finish a given task, we recorded their time as 15 minutes (the maximum allowed for each task). We found students in the DD display condition took longer to finish Task 1 ($M = 10.99$; $Med = 12.33$) than those in the on-demand condition ($M = 8.98$; $Med = 9.42$), but a Mann-Whitney U test[4] shows the difference was not significant, with a medium effect size ($p = 0.24$; Cohen's $d = -0.47$). Interestingly, we found that time taken to finish Task 2 (without hints) showed the reverse: students in the on-demand condition spent longer ($M = 10.55$; $Med = 12.23$) than those in the DD display condition ($M = 8.94$; $Med = 7.38$), but the difference is not significant ($p = 0.41$; Cohen's $d = 0.33$).

Based on these findings, we calculated how much students' performance *improved* from Task 1 (with hints) to Task 2 (without hints): Task 2 time - Task 1 time. We found that students in the DD display condition decreased their time from Task 1 to Task 2 ($M = -2.03$; $Med = -2.57$), while those in the on-demand condition increased their time ($M = 1.57$; $Med = 2.16$), and A Mann-Whitney U test shows the difference was significant, with a large effect size ($p = 0.02$; Cohen's $d = 0.85$). Together, these results suggest that on-demand display helps students more on the problem where they are available (short-term performance), but the DD display seems more likely to improve later performance without hints (i.e. learning). However, we are cautious in interpreting this result, since, as stated above, the overall impact of the DD display on students' Task 2 performance was positive but not significant.

One possible explanation for this difference would be that students in the on-demand condition simply asked for *more* hints on Task 1, improving their performance on Task 1 but not Task 2. However, we found that the students in the DD condition actually opened *more* hints ($M = 3.06$; $Med = 3$) than the on-demand condition ($M = 1.8$; $Med = 1$), and the difference was significant ($p = 0.04$; Cohen's $d = 0.59$). We also found that students *interacted* with hints differently in the two conditions. Students in DD display kept hints open for significantly more time ($M = 21.10$; $Med = 19$) than students in the on-demand condition ($M = 17.70$; $Med = 10$). A Mann-Whitney U test shows this difference is significant ($p < 0.01$; Cohen's $d = 0.23$). These results suggest that providing hints *only* at times when students are assumed to need them, allows them to process hints better [27], which affected their performance in Task 2. In addition, these results confirmed our hypothesis that timing the hints on students' needs can eliminate help abuse categories.

RQ2c: *What are students' perceptions of the helpfulness of both hint displays?* As Kardan et al., noted "Success of any adaptive support mechanism highly depends on the users' perception of its quality" [20], and measuring students' perceptions is lacking in most help-seeking literature. It seems possible that students would not perceive the DD display to be as helpful as the on-demand one, since it does not give them as much control over when they receive hints. To measure this, we asked students in the Post Task 1 survey "On a scale from 1 to 10, how helpful was iSnap overall?". We found that students in the DD group rated iSnap at least as helpful ($M = 7.36$; $Med = 8$) as students in the on-demand condition ($Mean = 6.23$; $Med = 7.5$). This suggests no evidence that deploying DD design in the programming environment was perceived as less helpful than that with traditional on-demand help.

Moreover, when we asked students in the DD group to elaborate on why they gave this rating, students noted that hints just popped up at the right time "every time I got lost there was a hint." [P1]. Other students felt the hints were adaptive to their code "It seemed to let me try to figure things out for a minute then tell me when I couldn't get it." [P7]. In Post Task 2 survey, when we asked students if hints they received in Task 1 helped in Task 2, 92% of the students noted that the hints helped them to progress in Task 2 because "the code [in Task 2] required me to use similar blocks that I had learned in the past task." [P5], and others mentioned specific concepts they had learned from the hints, e.g. "when the hints from task 1 explained how I have to rotate my pen at certain times, it helped me do the same during task 2. [P4]". Also, we did not find any comment that showed that students needed more help, which suggests that using the DD display, students did not notice their indirect lack of control on requesting help at any time.

4.4 Limitations and Broader Implications

We note three limitations to our findings. First, some of our results were inconclusive due to the small sample size. Also, our tasks may have been too easy, since 26.6% of students completed Task 1 in 8 minutes (although they claimed having no programming experience). However, the goal of this work was to identify unproductive help-seeking behavior in programming and explore how the DD display can improve that behavior, and our results suggest potential for it to do so. Second, while the varying difficulty of the programming tasks might have affected the results, we found no significant difference in help-seeking or performance between students with different tasks. Lastly, we had no pre-test, and some of our results could be explained by our two conditions having different levels of prior knowledge, despite being randomly assigned.

Overall, in this paper, we have three main contributions: 1) a preliminary taxonomy of novices' unproductive programming help-seeking behavior, which inspired us to (2) develop a data-driven (DD) adaptive display for hints to improve students' help-seeking, and (3) we conducted a controlled pilot study to evaluate the impact of the DD display on students' help-seeking behavior. Our results show promise for how the DD display could improve students' help-seeking behavior and possibly their learning, without reducing the perceived usefulness of the system. Also, our DD display for hints can be deployed in other programming environments that log students data and provide any type of feedback. We do note that our DD design does require an autograder for testing student's code in real time to monitor their progress through the assignment, but these are commonly available in programming environments [7, 18, 44]. Our system has implications for any classroom, whether or not automated help is available, since a similar system could also remind students to ask for help from instructors, peers, or message boards. In future work, we plan to replicate Study 2 on a large-scale population to verify the effect of DD display on performance and learning, and further explore the relation between help-seeking and different feedback types.

[4]We used non-parametric tests because the data was not normally distributed.

REFERENCES

[1] Vincent Aleven and Kenneth R Koedinger. 2001. Investigations into help seeking and learning with a cognitive tutor. In *Papers of the AIED-2001 workshop on help provision and help seeking in interactive learning environments*. 47–58.

[2] Vincent Aleven, Bruce Mclaren, Ido Roll, and Kenneth Koedinger. 2006. Toward meta-cognitive tutoring: A model of help seeking with a Cognitive Tutor. *International Journal of Artificial Intelligence in Education* 16, 2 (2006), 101–128.

[3] 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), 205–223.

[4] Vincent Aleven, Elmar Stahl, Silke Schworm, Frank Fischer, and Raven Wallace. 2003. Help seeking and help design in interactive learning environments. *Review of educational research* 73, 3 (2003), 277–320.

[5] Computing Research Association et al. 2017. Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006.(2017).

[6] Ryan Shaun Baker, Albert T Corbett, and Kenneth R Koedinger. 2004. Detecting student misuse of intelligent tutoring systems. In *International conference on intelligent tutoring systems*. Springer, 531–540.

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

[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] Susan Bergin, Ronan Reilly, and Desmond Traynor. 2005. Examining the role of self-regulated learning on introductory programming performance. In *Proceedings of the first international workshop on Computing education research*. ACM, 81–86.

[10] Piroska Biró and Maria Csernoch. 2014. Deep and surface metacognitive processes in non-traditional programming tasks. In *2014 5th IEEE Conference on Cognitive Infocommunications (CogInfoCom)*. IEEE, 49–54.

[11] Albert T 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 factors in computing systems*. ACM, 245–252.

[12] National Research Council et al. 2000. *How people learn: Brain, mind, experience, and school: Expanded edition*. National Academies Press.

[13] Ryan SJ d Baker, Sujith M Gowda, and Albert T Corbett. 2011. Towards predicting future transfer of learning. In *International Conference on Artificial Intelligence in Education*. Springer, 23–30.

[14] 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.

[15] 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.

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

[17] Luke Gusukuma, Austin Cory Bart, Dennis Kafura, and Jeremy Ernst. 2018. Misconception-Driven Feedback. *Proceedings of the 2018 ACM Conference on International Computing Education Research - ICER '18* 1 (2018), 160–168.

[18] 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, 223–227.

[19] Stuart A Karabenick and Richard S Newman. 2009. Seeking help: Generalizable self-regulatory process and social-cultural barometer. (2009).

[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, 3671–3680.

[21] Andrew J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 199–206.

[22] 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.

[23] 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.

[24] 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.

[25] Kazuhisa Miwa, Hitoshi Terai, Nana Kanzaki, and Ryuichi Nakaike. 2013. Stoic Behavior in Hint Seeking when Learning using an Intelligent Tutoring System. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 35.

[26] R Charles Murray and Kurt VanLehn. 2005. Effects of Dissuading Unnecessary Help Requests While Providing Proactive Help.. In *AIED*. Citeseer, 887–889.

[27] R Charles Murray and Kurt VanLehn. 2006. A comparison of decision-theoretic, fixed-policy and random tutorial action selection. In *International Conference on Intelligent Tutoring Systems*. Springer, 114–123.

[28] Mitchell J Nathan, Kenneth R Koedinger, Martha W Alibali, et al. 2001. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In *Proceedings of the third international conference on cognitive science*, Vol. 644648.

[29] Sharon Nelson-Le Gall. 1981. Help-seeking: An understudied problem-solving skill in children. *Developmental Review* 1, 3 (1981), 224–246.

[30] Sharon Nelson-Le Gall. 1986. *Help-seeking behavior in learning*. ERIC Clearinghouse.

[31] Richard S Newman. 1994. Adaptive help seeking: A strategy of self-regulated learning. (1994).

[32] Stellan Ohlsson, Barbara Di Eugenio, Bettina Chow, Davide Fossati, Xin Lu, and Trina C. Kershaw. 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.

[33] 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*.

[34] Thomas W. Price, Zhongxiu Liu, Veronica Cateté, and Tiffany Barnes. 2017. Factors Influencing Students' Help-Seeking Behavior while Programming with Human and Computer Tutors. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ACM, 127–135.

[35] Minna Puustinen. 1998. Help-seeking behavior in a problem-solving situation: Development of self-regulation. *European Journal of Psychology of education* 13, 2 (1998), 271.

[36] Leena Razzaq and Neil T Heffernan. 2010. Hints: is it better to give or wait to be asked?. In *International Conference on Intelligent Tutoring Systems*. Springer, 349–358.

[37] Leena Razzaq, Neil T Heffernan, and RW Lindeman. 2007. What level of tutor feedback is best. In *Proceedings of the 13th Conference on Artificial Intelligence in Education, IOS Press*.

[38] A Renkl. 2002. Learning from Worked-out Examples: Instrucfional Explanafions Supplement Self Explanafions. *Learning and Instrucfion* 12 (2002), 149–176.

[39] 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.

[40] Ido Roll, Ryan SJ d Baker, Vincent Aleven, and Kenneth R Koedinger. 2014. On the benefits of seeking (and avoiding) help in online problem-solving environments. *Journal of the Learning Sciences* 23, 4 (2014), 537–560.

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

[42] Bram E Vaessen, Frans J Prins, and Johan Jeuring. 2014. University students' achievement goals and help-seeking strategies in an intelligent tutoring system. *Computers & Education* 72 (2014), 196–208.

[43] Kurt Vanlehn, Collin Lynch, Kay Schulze, Joel A Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. 2005. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education* 15, 3 (2005), 147–204.

[44] 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)*.

[45] H Wood and D Wood. 1999. Help seeking, learning and contingent tutoring. *Computers & Education* 33, 2-3 (1999), 153–169.