The Impact of Adding Textual Explanations to Next-step Hints in a Novice Programming Environment

Samiha Marwan
Nicholas Lytle
North Carolina State University
Raleigh, North Carolina
{samarwan,nalytle}@ncsu.edu

Joseph Jay Williams
University of Toronto
Toronto, Canada
williams@cs.toronto.edu

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

ABSTRACT

Automated hints, a powerful feature of many programming environments, have been shown to improve students' performance and learning. New methods for generating these hints use historical data, allowing them to scale easily to new classrooms and contexts. These scalable methods often generate next-step, code hints that suggest a single edit for the student to make to their code. However, while these code hints tell the student what to do, they do not explain why, which can make these hints hard to interpret and decrease students' trust in their helpfulness. In this work, we augmented code hints by adding adaptive, textual explanations in a block-based, novice programming environment. We evaluated their impact in two controlled studies with novice learners to investigate how our results generalize to different populations. We measured the impact of textual explanations on novices' programming performance. We also used quantitative analysis of log data, self-explanation prompts, and frequent feedback surveys to evaluate novices' understanding and perception of the hints throughout the learning process. Our results showed that novices perceived hints with explanations as significantly more relevant and interpretable than those without explanations, and were also better able to connect these hints to their code and the assignment. However, we found little difference in novices' performance. Our results suggest that explanations have the potential to make code hints more useful, but it is unclear whether this translates into better overall performance and learning.

KEYWORDS

Intelligent tutoring systems, next step hints, textual hints, computer science education

ACM Reference Format:
Samiha Marwan, Nicholas Lytle, Joseph Jay Williams, and Thomas Price. 2019. The Impact of Adding Textual Explanations to Next-step Hints in a Novice Programming Environment. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland UK.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3304221.3319759

ITiCSE '19, July 15–17, 2019, Aberdeen, Scotland UK

ACM ISBN 978-1-4503-6301-3/19/07…$15.00
https://doi.org/10.1145/3304221.3319759

1 INTRODUCTION

Introductory programming courses often have high attrition and failure rates, with pass rates estimated as low as 68% [6, 38]. As such, there is a growing evidence to suggest that novice programming students need additional support [7, 37]. This is especially true in large classrooms and MOOCs, where instructors cannot interact with every student [17, 21]. To address this, researchers have developed intelligent systems for automatically generating support for students during programming [2, 13, 16, 31], such as hints and feedback, which have been shown to improve students' learning outcomes [8, 9]. These systems often offer next step, edit-based hints, which suggest an edit that a student should make to bring their code closer to a correct solution [10, 28]. Data-driven systems can generate these next-step hints automatically using historical students' data [31, 34], rather than with expensive expert models (e.g. [24]), allowing them to scale easily to new classrooms and contexts. These next-step hints are powerful as they can support students *during* program construction, and *automatically* adapt to the student's current code to support different student solutions. However, these data-driven, next-step hints generally show only *what* to do, not *why*. Previous work has shown this lack of explanations can make next-step hints difficult for novices to interpret and connect to their current code [29], leading to a decrease in how often students request and follow hints, and eroding their trust in the system [29, 32].

In this paper we introduce a straightforward method for generating textual explanations to accompany automated, next-step programming hints. We implement this feature in iSnap [28], an extension of the block-based *Snap!* programming environment [12, 14], which offers on-demand, data-driven, next-step hints. These hints are generated adaptively to respond specifically to a student's current code. We refer to these hints as *code hints*, since they show only the code that should be changed. We designed the textual explanations to connect these code hints to the programming exercise objectives and explain the functionality of the suggested blocks. Previous work suggests that these explanations could improve novices' programming outcomes, but that novices do not always read such explanations [11]. We therefore evaluated the impact of adding these textual explanations to code hints in iSnap through two controlled experiments with different populations.

In Experiment 1, we conducted a controlled study on novices in an introductory programming course for non-CS majors. We studied the effect of adding textual explanations to code hints, and found promising trends across multiple exercises, suggesting that explanations may increase students' willingness to use and follow

the hints. However, as in previous work [1, 28], the majority of students did not use hints, so our sample size was small, and our results were inconclusive. To investigate further, we conducted a second experiment with crowd workers recruited on Amazon's Mechanical Turk platform, which has been suggested as an appropriate alternative to university participants [5, 19]. This experiment allowed us to recruit a larger number of learners, collect fine-grained survey data, and give learners different, uneven levels of support – all of which were infeasible in our first classroom experiment. In addition to analyzing learners' final outcomes, as in previous work [35], we make empirical analysis using log data, self-explanation prompts, and frequent feedback surveys to understand learners' understanding and perception of the hints throughout the learning process. Both experiments studied novices with little-to-no programming experience. We find learners who received code hints *with* textual explanations rate these hints as significantly more useful, and were more likely to follow these hints which shows similar trend to Experiment 1. In addition, learners who received textual explanations were also significantly more likely to explain the relationship between the received hints, their code and the assignment objectives. However, we found no significant difference in the two groups' programming performance.

In Experiment 1, we investigated 2 research questions: *How does adding textual explanations to code hints impact (1) The perceived usefulness of hints? (2) The hints' request rate?*. In Experiment 2, due to the fewer restrictions in this study, we collected surveys to better investigate RQ1 and investigated 2 more research questions: *How does adding textual explanations to code hints impact (3) Learners' performance on the current and future programming assignments? (4) Learners' ability to self-explain the relationships between the code hints to their own code and the assignment?*

In summary, the contributions of this work are: (1) A system that combines the scalability of adaptive, next-step programming "code hints" with textual explanations[1]. (2) An exploratory study suggesting that these explanations may increase students' willingness to follow code hints. (3) Evidence suggesting that adding these textual explanations to next-step hints increases their perceived usefulness, and learners' ability to explain the purpose of the hints.

2 RELATED WORK

Many educational programming systems provide automated, contextualized support to novices during problem solving. This support can greatly improve learning outcomes [11]. For example, Corbett et al. [9] showed that students who received various forms of feedback while using the system completed subsequent programming assessment in significantly less time with significantly fewer errors. Two common means of this automated support are next-step hints [25, 28, 31] and feedback hints [13, 23]. Next-step hints suggest a specific edit that a student should make to their code to make progress towards a solution, which can be presented through textual instructions [34] or a visual demonstration of the suggested edit [28] (see Figure 1). Automated, next-step hint generation is a rapidly growing area of research, and researchers have developed a variety of techniques, including data-driven approaches [27, 31, 34], program synthesis [26, 36], and automated program repair [39]. Prior evaluations of next-step programming hints suggest that they may increase students' immediate performance, but the results were inconclusive due to low rates of hint-usage [32, 33]. This low rates of hint-usage might be due to the hints' lack of explanation and interpretability [28, 29].

Feedback hints offer students expert-authored help messages, often in response to specific errors in student code. For example, SQLTutor [23] and INCOM [20] are tutoring systems that use a constraint-based model to provide feedback hints to students whenever a constraint is violated. Unlike next-step hints, the feedback provided by these systems usually comes after students submit their code. However, in an evaluation of the Lambda Autograder [3], which offers such feedback, students mentioned they prefer feedback for each step rather than general feedback at the end. In [15], Head et al. proposed a system that allows teachers to write reusable feedback for programming exercises in Python. In this study, teachers expressed how useful the system was in saving time in grading. However, they mentioned that some students might still need more individualized feedback on their code. In this work, both textual explanations and code hints are adaptive to students' code and can be provided at any time during programming.

One challenge with evaluating automated support, such as next-step hints and feedback messages, is that these supports are usually part of larger programming systems, and researchers often evaluate these systems as a whole, without singling out specific support features to better understand them [18]. For example, Gerdes et al. evaluated their programming tutor, Ask-Elle [13], and its adaptive support through questionnaires. This methodology may be insufficient at analyzing the real impact of these hints on student's behavior during programming because they focus more on student's *opinion* and not their actual *behavior*. In this study, we isolate the impact of one specific element of support – textual explanations – to evaluate it directly. Another common way of evaluating these systems in literature is by checking students' performance when solving exercises with support [16, 28]. However, improved performance with support does not necessarily imply improved learning, since hints and feedback often give the student part of the correct solution [4]. In this paper, we address these issues in two ways: (1) by evaluating student performance on future tasks as a measure of learning (2) by using self-explanations as an alternative technique to measure the impact of hints on students' knowledge [22].

3 DESIGN OF AUTOMATED TEXTUAL EXPLANATIONS

We designed automated textual explanations hints for iSnap, a block-based novice programming environment that provides students with on-demand, data-driven *code hints* during programming [28]. iSnap extends *Snap!*, which is used by thousands of students in "The Beauty and Joy of Computing" AP Computer Science Principles classes [12]. iSnap's hints are generated by the SourceCheck algorithm that uses a database of correct student solutions for a given problem [31]. SourceCheck matches the current code of a hint-requesting student to the nearest correct solution in the database and suggests edits that will bring the student closer to that solution. Students request hints on iSnap by clicking a *Check My Work* button, after which their code is annotated with multiple hint

[1] iSnap's demo, datasets and source code are available at http://go.ncsu.edu/isnap.

buttons, each corresponding to a contextual hint generated based on their current code [28]. When a student clicks on the hint button, a hint window is shown suggesting a single edit to students' code, such as inserting a code element as shown in Figure 1.

Each of SourceCheck's next-step hints suggest a code block that students can add. That block comes from a correct solution in SourceCheck's database, corresponding to a node in that solution's abstract syntax tree (AST) [31]. We generate textual explanations for a given problem in iSnap by first identifying all common AST nodes in the database of solutions, such as the "pen down" block shown in Figure 1. We then manually annotate each of these solution AST nodes with a textual explanation for the corresponding hint. Note that in the experiments presented in this paper, we used expert-authored solutions, rather than student solutions, to generate the hints, as this has been shown to produce higher-quality hints [32], but our approach could be applied to either.

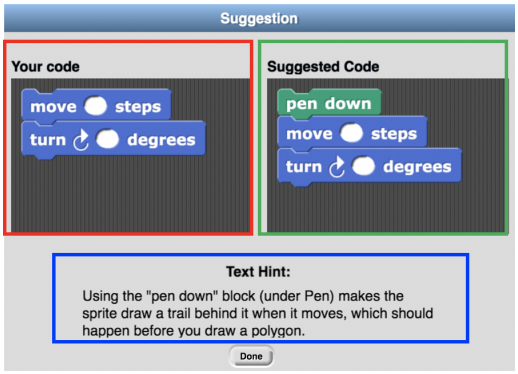

Figure 1: A next-step hint with textual explanation in iSnap. The student's code is shown in the top-left (red box), with suggested code hint in the top-right (green box), annotated with a textual explanations (blue box).

In a programming exercise that asks students to draw a polygon, if a code hint is shown visualizing the need to use the "pen down" block (needed for drawing), the textual explanation in this case will be: *"Using the "pen down" block under Pen, makes the sprite draws a trail behind it when it moves, which should happen before you draw a polygon"* as shown in Figure 1. Our textual explanations were authored to provide 3 types of information to the student: (1) where can they find the relevant block *"Using the pen down block under Pen"*; (2) the functionality of this block *"makes the sprite draws a trail behind it when it moves"*; and (3) why this block should be used in the correct solution *"which should happen before you draw a polygon"*. We related the functionality of the block to the assignment objectives, as our prior work suggested this was a point of difficulty for many students in understanding hints [29]. For the experiments described in this work, the last author, who is familiar with the assignments and classroom context, wrote the textual explanations, including the three types of information described above.

4 EXPERIMENT 1: CLASSROOM STUDY

In our first, exploratory experiment, we investigated the impact of adding textual explanations to code hints in a classroom context. We investigated our research questions by focusing on how explanations impacted students' willingness to request and follow hints. Previous work has shown that this is a good proxy for perceived hint quality and usefulness [32].

4.1 Population and Procedure

The study was conducted during an introduction to programming course for non-majors in a research university during Fall 2018. Before the first exercise, a researcher explained iSnap's hint interface to students and encouraged them to use it whenever they needed support. There was no limit to the number of hints students could request, and iSnap reminded students about the hint feature at the beginning of each exercise. Our experiment was conducted on 2 in-lab exercises, where students could request help from undergraduate teaching assistants (TAs), and on 1 homework exercise, completed independently. Since students could request help from TAs during in-lab assignments and almost all complete the homework correctly, we do not analyze the quality of their programming performance in this experiment (though we do so in Experiment 2). The total number of students was 45, and all students completed their work using iSnap.

The students were randomly assigned into two groups: Group A (G_A) and Group B (G_B). Our experiment has a crossover design, so that both groups had access to textual explanations at different times. For the first two tasks, G_A received code hints with textual explanations, while G_B received only code hints, but for the third task[2], G_A was given only code hints, while G_B was given both textual explanations and code hints. As shown in Table 1, only 15-21 students requested hints on any assignment.

4.2 Measures and Results

We used 2 measures to understand the impact of textual explanations on the perceived usefulness and quality of hints. We first looked at the number of hints requested by students in each condition. As shown in the "H" columns of Table 1, there was no consistent trend across exercises. Students who *did* ask for hints in both groups seemed to ask for similar numbers of hints. The second measure we looked at was the hint follow rate (FR), or the average number of hints followed by each student. We defined a hint as "followed" if a student created the *suggested* block in that hint before receiving a new one. As shown in the "FR" columns of Table 1, the group with textual explanations had a higher average follow rate across assignments (approaching 100% in the homework exercise), ranging from 12 to 25 percentage points. These results suggest that students who received textual explanations with their hints were more convinced to follow the hints.

Looking more closely at *which* hints were followed and not, we can see how explanations may impact students' follow rate. Remember, for any assignment, there are a fixed number of unique, possible hints (each corresponding to one code block). When shown to a student, these hints are adapted to their current code (i.e. showing where to add the block). There were 9, 10 and 14 unique possible hints on the three exercises, respectively. A given hint may be shown to different students, and a student may view the same hint

[2]This was also true for a fourth task, but due to a power outage, we were unable to collect data for this task.

Table 1: Experiment 1 Results: For each exercise, the number of students who requested any hints (n), the mean number of hints requested (H), and the mean hint follow rate (FR). $*$ indicates the group which received textual explanations.

	Lab 1			Homework 1			Lab 2			
	n	H	FR	n	H	FR		n	H	FR
G_A^*	11	9.27	0.53	7	5.7	0.95	G_B^*	5	10	0.70
G_B	10	7.27	0.56	8	6.25	0.70	G_A	10	9.90	0.47

multiple times. We say a hint has been *ignored* if at least one student received it and chose not to follow it. For students who received code hints *only*, the number of unique, ignored hints were 7/9, 6/10, 8/14 for the 3 exercises respectively. However, for students who also received textual explanations, the number of unique, ignored hints were 3/9, 1/10, and 5/14, respectively. This suggests that while many different kinds of code hints were ignored by students, when these hints are accompanied by explanations, only a few hints are ever ignored. We hypothesized that these few ignored hints may have had inadequate explanations, which were confusing or hard to act on. For example, in the second exercise the only hint that was not followed by G_A was *"Each time your code repeats, your sprite should draw one side of the polygon"*, which explains the code hint that suggested the addition of the *"move forward"* block.

4.3 Discussion

Our results suggest that textual explanations can serve their intended purpose of improving students' ability to understand and follow hints. Students with textual explanation were more likely to follow the hints they received. With explanations, most hints were followed anytime they were seen, which was not true for hints without explanations. While this exploratory analysis suggests some promise for textual explanations, our data consists of only a few students. Though trends were consistent across 3 exercises, we cannot make strong conclusions from these results. One limitation of this experiment is that students were allowed to request hints on demand, and, as shown in previous studies, students who most need support often do not ask for hints when they need them [1, 28]. We therefore can only study a subset of the students of interest. It is worth noting that the first and third exercises were in-lab, where students might have had additional support from instructors, possibly affecting how they used hints.

To get a better understanding of why some hints were not followed, we manually inspected each hint that was not. We found two potential reasons. First, some students requested a hint and then immediately closed it and requested another (without doing any additional edits to their code), repeating this process until either they stopped requesting hints or followed one. This is possible because iSnap displays multiple hint suggestions at a time for different parts of a student's code. This behavior suggests that sometimes students were looking for help in a specific part of their code and ignored hints for other parts, resulting in a decreased follow rate. The second reason, as noted earlier, is that some of the unfollowed hints with textual explanations may not have had the clearest explanations. In our future studies we plan to rephrase them or add the option of having more than one textual explanation for the same code hint in the hope of making these explanations more understandable and effective for novices.

5 EXPERIMENT 2: CROWD WORKERS

Based on promising but inconclusive results of Experiment 1, we designed Experiment 2 to overcome some of its limitations by recruiting larger population, providing hints pro-actively (instead of on-demand) and collecting surveys and self-explanations on the hints to better understand their impact on learners' understanding and their perceived usefulness. Experiment 2 aimed to answer the following research questions: How does adding textual explanations to code hints impact: (1) Learners' perceived usefulness of the hints? (2) Learners' performance on the current and future programming assignments? (3) Learners' ability to self-explain the relationships between code hints, their own code and the assignment?

5.1 Population and Procedure

In this experiment, our learners consisted of crowd workers who were recruited from Amazon's Mechanical Turk platform which has been discussed to be as effective form of conducting large-scale user studies as using university participants [5, 19]. All learners reported that they had no prior programming experience, though we did not collect any further demographic information. We paid learners $4-7 to complete the study (varying the amount to increase speed of recruiting). The data were collected as part of a larger study on the effect of different types of hint support which included 209 learners. We focus on a subset of that data relevant to our research questions about the impact of adding textual explanations to code hints, which included just 92 learners. We also excluded 4 of them from our study: 2 learners did not write any code, 1 learner finished the exercise before receiving the first hint and the last one gave meaningless self-explanations (discussed below), leaving a total of 88 learners in our analysis.

During the study, learners completed a programming task (detailed below), during which they were provided with some form of automated help. All 88 learners received code hints. Of these, 48 were randomly assigned to receive textual explanations with their code hints (group G_T), while the other 40 received only code hints (group G_C). Additionally, approximately half of the participants were randomly assigned to self-explain each received hint, and we use this data in our evaluation, as explained below. Learners started the study by reading a short tutorial on block-based programming in iSnap for 5 minutes[3]. The tutorial explained how to use iSnap's user interface as well as all the programming concepts needed for the programming task. These concepts included input/output, loops, operators and drawing shapes, which were all explained by text and short animated videos. Afterwards, learners started to work on a programming task for 15 minutes. The programming task involved drawing a polygon based on the number of sides the user inputs.

During programming, iSnap interrupted the learners every two minutes and provided them a hint according to their condition (e.g. showing a code hint) because we wanted rich data about multiple hints, rather than biased sample when learners request it and to avoid the limitations in Experiment 1 as well. After receiving the

[3]We provided only a short time for the tutorial, since our goal was to study how students learned hints while *programming*.

hint, iSnap asked them for their thoughts on the action (Post-help survey). The 2 minute action timer restarted after learners finished the post-help survey, giving them 2 more minutes of work before being interrupted again. The 15 minute timer was not paused during these surveys, though they were identical across conditions. While offering help proactively, as we did in this study, is less common than offering on-demand help, however, results from a pre-help survey (not analyzed in this paper) showed that learners desired help the vast majority of the time when it was given. After 15 minutes, learners were stopped and then worked on a second programming task (Task 2) for 15 minutes. Task 2 asked them to create a program that draws a strip of triangles, with the number of triangles given by the user. This task uses similar programming concepts as Task 1, but is slightly more advanced. Unlike Task 1, every 2 minutes learners were given a random hint independent of their condition in Task 1. We found that each learner received 2-5 hints during each task, varying depending on how long they took on the pre-help and post-help surveys, and whether there was a reasonable hint.

5.2 Measures

We collected 3 types of data relevant to this analysis: log data, post-help survey responses, and self-explanations. To measure learners' performance, we divided each programming task into 4 objectives (e.g. "ask the user for input and use it correctly", "draw a shape," etc.). Finishing all the objectives correctly is equivalent to successful completion of the task. We developed an automatic grader to determine which objectives learners have completed, and manually verified these grades for approximately half of participants. After each hint, iSnap asked learners 4 questions in the Post-help survey to rate the support they received (e.g. code hints). These 4 questions measured: (1) Usefulness: *How useful were the action(s) that iSnap just took?*, (2) Relevance: *How well did this hint address what you were working on or stuck on?*, (3) Progress: *How well did this hint help you progress towards a correct solution?*, (4) Interpretability: *How easily could you interpret the meaning of this hint?*. We based questions 2-4 on a rubric for rating hint quality [32]. In addition, we measured the hints' *follow rate*, which is the average of hints followed by each learner, by auto-inspecting learners' log traces to know if they really have followed the hint or not.

Approximately half of participants (19 in G_T and 21 in G_C) received a self-explanation prompt after each hint with a randomized prompt (e.g. *"How would you use this hint?"*, *"Why do you think Snap recommended this hint?"*). We used their responses to measure learners' ability to explain the purpose of the hints, evaluating a total of 125 self-explanations. Two researchers familiar with the task used a yes/no rubric to independently assess whether learners' self-explanations connected their hints to their: (1) Current Code: *Is the explanation related to the learner's current code situation?*; (2) Programming Concepts: *Does the explanation show that the learner has learned anything related to programming concepts?*; (3) Assignment: *Does the explanation show how the hint can help the learner in his assignment?*. The two researchers blindly rated learners' self-explanations, i.e. without knowing the learner's condition (e.g. received code hints). The two researchers divided the first half of the self-explanations answers into 3 rounds for rating each of the 3 questions. After each round, conflicts were discussed and resolved. We calculated Cohen's kappas for each question in each round, where 9 kappas ranged from 0.724 to 1, indicating substantial agreement. Afterwards, the second half of the self-explanations' answers was divided between the two researchers for grading.

5.3 Results

Hints' perceived usefulness and follow rate. We averaged learners' responses in the Post-help Survey across Task 1 to assign one rating per question to each learner. As shown in Table 2, we compared average ratings between G_T learners and G_C learners. We used Mann-Whitney U tests here, as in the remainder of this section, as our data were non-normal. We found that G_T learners rated iSnap's help as significantly more useful ($p = 0.021$), relevant ($p = 0.030$) and interpretable ($p = 0.018$) than G_C. Learners in G_T also rated the hints as more effective in helping them to make progress; however, the rating differences were not significant ($p = 0.109$). In addition, we found that the learners' hint follow rate was higher in G_T than G_C, as in Experiment 1. However, despite a medium effect size (Cohen's $d = 0.38$) this difference was not significant ($p = 0.137$).

Learners' Performance. We compared number of objectives that learners completed during Task1 (with hints determined by their condition) and in Task2 (with randomized help, independent of their condition). As shown in Table 2, we found no significant difference in the number of objectives completed by each group in Task1 ($p = 0.66$) or in Task2 ($p = 0.28$). One simple way we hypothesized the textual explanations might improve learners' performance is by helping students to locate code blocks, since this information was included in the explanation. We identified all hints that asked the learner to find and create a new code block, where the learner correctly followed this hint (before receiving another hint). For each learner, we calculated the average time they took to create these blocks. However, the difference in this average time for learners in G_C (Med = 28.1) and G_T (Med=24.5) was not significant ($p = 0.57$).

Learners ability to self-explain the relationships between code hints, their own code and the exercise's objectives. For learners who received self-explanation prompts after receiving hints, we rated their self-explanations, as described in Section 5.2. For each learner, we determined the percentage of their self-explanations which connected the received hint to : 1) their current code (Current Code), 2) correct programming concept(s) (Programming Concepts), and 3) the objective(s) of the programming assignment (Assignment). As shown in Table 3, we found that these percentages were much higher in G_T than G_C. A Mann-Whitney U Test shows that this difference is statistically significant for the first and third questions ($p = 0.020$, $p = 0.057$, $p = 0.023$).

5.4 Discussion

Our results suggest that textual explanations can improve code hints by increasing how useful, relevant and interpretable learners perceive them to be (RQ1) and learners' ability to self-explain the hints (RQ3). There is also some evidence that the explanations increase learners' willingness to follow hints, as suggested by Experiment 1, though the difference was not significant. This suggests that, despite concerns raised in prior work [11], learners did in fact

Table 2: Experiment 2 Results. The table gives Mean (M), Standard Deviation (SD) and Median (Med) for both groups, with the p value for comparison using a Mann-Whitney U test, and effect size, Cohen's d.

	Interpretability			Relevance			Progress			Usefulness			Follow Rate			Task 1 Perf.			Task 2 Perf.		
	M	SD	Med	M	SD	Med	M	SD	Med	M	SD	Med	M	SD	Med	M	SD	Med	M	SD	Med
$G_T(48)$	8.24	2.24	9	8.31	2.12	9	7.90	2.59	9	8.28	1.91	9	0.75	0.31	0.87	2.31	1.32	2	2.15	1.46	2
$G_C(40)$	7.53	1.82	7.22	7.46	2.13	8	7.47	1.97	7.1	7.36	2.21	7.91	0.62	0.39	0.7	2.15	1.52	2	2.48	1.35	2
p	**0.018**			**0.03**			0.109			**0.021**			0.137			0.665			0.288		
d	0.34			0.39			0.18			0.44			0.38			0.11			0.23		

Table 3: Self-Explanations Results in Experiment 2. For each question, Mean (M), Standard Deviation (SD) and Median (Med) with the p value for comparison using a Mann-Whitney U test, and effect size, Cohen's d.

	Current Code			Prog. Concepts			Assignment		
	M	SD	Med	M	SD	Med	M	SD	Med
$G_T(19)$	0.52	0.42	0.5	0.29	0.34	0.25	0.35	0.35	0.33
$G_C(21)$	0.22	0.28	0	0.12	0.27	0	0.12	0.27	0
p	**0.020**			0.057			**0.023**		
d	0.852			0.55			0.73		

read explanations and derive some benefits. Interestingly, despite these advantages, there is no evidence that textual explanations improved learners' performance on the current or future tasks (RQ2). In prior analysis of this data, we did find that learners with code hints (without textual explanations) performed better on Task 1 than those with no hints [30]. This suggests that code hints do improve learners' immediate performance, but our results indicate that the textual explanations did not contribute to this improvement. It is interesting that learners' increased ability to relate hints to programming concepts suggests a type of learning, but this did not translate into improved performance. This may be because, even with textual explanations, learners still only related 29% of hints they received to programming concepts on average.

Regarding the design of the textual explanations, recall that our goal was to provide the learner with 3 types of information: block's functionality, why it is needed in the given assignment, and the location of the block. Our analysis on learners' self-explanations show that this information helped learners to relate hints to their own code, programming concepts (e.g. definition of loops, etc.) and to the assignment objectives. For example, one learner in G_T received a hint to use the "repeat block", then they were asked: *"How would you use this hint?"*. The learner self-explained the hint *"I would use it to repeat however many sides there are in the polygon"*. The third type of information was the location of the suggested block. However, we found no evidence that this information helped learners in G_T to find blocks faster than those in G_C. This might be because iSnap blocks are already color-coded, so when a learner is given a code hint he can find its location by matching its color with the category of the same color. As a result it is possible that learners did not need this feature to be stated explicitly in text. These results show that our explanations design fulfilled 2 of its objectives.

In this Experiment, there are several limitations to our results. First, the population consists of paid crowd workers with no prior programming experience. Their motivations and prior knowledge may differ from those of other populations of learners where programming hints are used. However, this study allowed us to be flexible with the design and add as many constraints (like self-explanations and post-help surveys) which is usually hard to do in a classroom setting, like in Experiment 1, that is restricted to a specific time and format. Second, the 2-minute counter we used to time when participants received hints is different from the on-demand way that many hints are presented [13, 34], and may have been distracting for users. However, prior work shows that students who most need support often do not ask for hints when they need them [1, 28]. Our approach of offering hints proactively allowed us to study hints' impact on all learners, not just the ones who are willing to ask for help. We also note that the textual explanations were still visible to learners when they were writing self-explanations of hints. It is unclear to what extent these explanations reflect students' *understanding* of the hints, as opposed to a summary of the visible explanations. However, we found no cases where a learner directly copied a textual explanation in their self-explanation.

6 CONCLUSIONS AND FUTURE WORK

In this paper we designed complementary textual explanations for our existing code hints in iSnap, with the goal of overcoming the limitations of code hints that only tell a student what to do. We evaluated our combined support by conducting two controlled studies on different populations. In our first classroom study, we found evidence that hint explanations may improve students' willingness to follow code hints, though we did not have enough participants to confirm these results. We conducted another study on a larger population of crowd workers, allowing us to do additional empirical analysis. Our results showed that learners who received textual explanations in addition to code hints perceived iSnap's support as significantly more useful, relevant and interpretable and had a better understanding of the hints provided than learners who received only code hints. We found no contradictions in the results of the two studies, and we argue that our results from the second study should generalize to a classroom context. In our future work, we will improve the design of textual explanations by supporting not only hints that suggest *to insert* a given block, but also those that suggest *to delete* a block as well. Additionally, a larger classroom study on more difficult exercises using a pre and post tests to measure learning gain as well could confirm how well our findings from Experiment 2 generalize to a classroom setting. Finally, we aim to design an interface for iSnap for teachers to author these explanations for new assignments.

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] John R. Anderson, Frederick G. Conrad, and Albert T. Corbett. 1989. Skill Acquisition and the LISP tutor. *Cognitive Science* 13, 4 (1989), 467–505.

[3] Michael Ball. 2018. *Lambda: An Autograder for Snap!.* Technical Report. Electrical Engineering and Computer Sciences University of California at Berkeley. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-2.pdf

[4] Joseph E Beck, Kai-min Chang, Jack Mostow, and Albert Corbett. 2008. Does help help? Introducing the Bayesian Evaluation and Assessment methodology. In *International Conference on Intelligent Tutoring Systems*. Springer, 383–394.

[5] Tara S. Behrend, David J. Sharek, Adam W. Meade, and Eric N. Wiebe. 2011. The viability of crowdsourcing for survey research. *Behavior Research Methods* 43, 3 (25 Mar 2011), 800.

[6] Jens Bennedsen and Michael E. Caspersen. 2007. Failure rates in introductory programming. *ACM SIGCSE Bulletin* 39, 2 (2007), 32. https://doi.org/10.1145/1272848.1272879

[7] Xianglei Chen and Matthew Soldner. 2013. *STEM Attrition: College Students' Paths Into and Out of STEM Fields.* Technical Report. National Center for Education Statistics, Institute of Education Sciences, U.S. Department of Education. http://nces.ed.gov/pubs2014/2014001rev.pdf

[8] Albert T. Corbett. 2001. Cognitive Computer Tutors: Solving the Two-Sigma Problem. In *Proceedings of the International Conference on User Modeling*. Springer, 137–147.

[9] 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.

[10] Davide Fossati, Barbara Di Eugenio, Stellan Ohlsson, Christopher Brown, and Lin Chen. 2010. Generating proactive feedback to help students stay on track. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6095 LNCS, PART 2 (2010), 315–317.

[11] 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.

[12] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The Beauty and Joy of Computing. *ACM Inroads* 6, 4 (2015), 71–79.

[13] Alex Gerdes, Bastiaan Heeren, Johan Jeuring, and L. Thomas van Binsbergen. 2017. Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback. *International Journal of Artificial Intelligence in Education* 27, 1 (2017), 1–36.

[14] Brian Harvey, Daniel Garcia, Josh Paley, and Luke Segars. 2012. Snap!:(build your own blocks). In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 662–662.

[15] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. In *Proceedings of the ACM Conference on Learning @ Scale*. ACM, 89–98.

[16] Jay Holland, Antonija Mitrovic, and Brent Martin. 2009. J-LATTE: a Constraint-based Tutor for Java. In *Proceedings of the International Conference on Computers in Education*. University of Canterbury. Computer Science and Software Engineering, 142–146.

[17] Shalini Kaleeswaran, Anirudh Santhiar, Aditya Kanade, and Sumit Gulwani. 2016. Semi-Supervised Verified Feedback Generation. *CoRR* (2016), 739–750.

[18] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2016. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16*. ACM, 41–46. https://doi.org/10.1145/2899415.2899422

[19] Aniket Kittur, Ed H Chi, and Bongwon Suh. 2008. Crowdsourcing user studies with Mechanical Turk. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 453–456.

[20] Nguyen-Thinh Le, Wolfgang Menzel, and Niels Pinkwart. 2009. Evaluation of a constraint-based homework assistance system for logic programming. *Proceedings of the 17th International Conference on Computers in Education* (2009), 51–58.

[21] Victor J. Marin, Tobin Pereira, Srinivas Sridharan, and Carlos R. Rivero. 2017. Automated personalized feedback in introductory Java programming MOOCs. *Proceedings - International Conference on Data Engineering* August (2017), 1259–1270.

[22] Danielle S. McNamara. 2017. Self-Explanation and Reading Strategy Training (SERT) Improves Low-Knowledge Students' Science Course Performance. *Discourse Processes* (2017).

[23] Antonija Mitrovic. 1998. A knowledge-based teaching system for SQL. In *Proceedings of ED-MEDIA*, Vol. 98. 1027–1032.

[24] Antonija Mitrovic, Brent Martin, and Pramuditha Suraweera. 2007. Intelligent tutors for all: Constraint-based modeling methodology, systems and authoring. *IEEE Intelligent Systems* 22 (2007), 38–45.

[25] Benjamin Paaßen, Barbara Hammer, Thomas William Price, Tiffany Barnes, Sebastian Gross, and Niels Pinkwart. 2017. The Continuous Hint Factory - Providing Hints in Vast and Sparsely Populated Edit Distance Spaces. *arXiv preprint arXiv:1708.06564* 10, 1 (2017), 1–35. arXiv:1708.06564

[26] Daniel Perelman, Sumit Gulwani, and Dan Grossman. 2014. Test-driven synthesis for automated feedback for introductory computer science assignments. *Proceedings of Data Mining for Educational Assessment and Feedback (ASSESS 2014)* (2014).

[27] C Piech, J Huang, A Nguyen, M Phulsuksombati, M Sahami, and L Guibas. 2015. Learning program embeddings to propagate feedback on student code. *arXiv preprint arXiv:1505.05969* (2015), 1093–1102. arXiv:arXiv:1505.05969v1

[28] Thomas W Price, Yihuan Dong, and Dragan Lipovac. 2017. iSnap: towards intelligent tutoring in novice programming environments. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 483–488.

[29] 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.

[30] Thomas W. Price, Joseph J. Williams, and Samiha Marwan. 2019. A Comparison of Two Designs for Automated Programming Hints. *2nd Educational Data Mining in Computer Science Education (CSEDM) Workshop at the International Conference on Learning Analytics and Knowledge (LAK)* (2019).

[31] Thomas W. Price, Rui Zhi, and Tiffany Barnes. 2017. Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming. In *Proceedings of the International Conference on Educational Data Mining*.

[32] Thomas W Price, Rui Zhi, and Tiffany Barnes. 2017. Hint generation under uncertainty: The effect of hint quality on help-seeking behavior. In *International Conference on Artificial Intelligence in Education*. Springer, 311–322.

[33] Kelly Rivers. 2017. *Automated Data-Driven Hint Generation for Learning Programming.* PhD. Carnegie Mellon University.

[34] 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.

[35] André L. Santos. 2012. An open-ended environment for teaching Java in context. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12*. ACM Press, New York, New York, USA, 87.

[36] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. 2013. Automated feedback generation for introductory programming assignments. *Acm Sigplan Notices* 48, 6 (2013), 15–26.

[37] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A systematic review of approaches for teaching introductory programming and their influence on success. *Proceedings of the tenth annual conference on International computing education research - ICER '14* (2014), 19–26.

[38] Christopher Watson and Frederick W B Li. 2014. Failure rates in introductory programming revisited. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*. ACM, ACM, 39–44.

[39] Jooyong Yi, Umair Z. Ahmed, Amey Karkare, Shin Hwei Tan, and Abhik Roychoudhury. 2017. A Feasibility Study of Using Automated Program Repair for Introductory Programming Assignments. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*. ACM, 740–751. https://doi.org/10.1145/3106237.3106262