2020

Effect of Implementing Subgoals in Code.org's Intro to Programming unit in Computer Science Principles

Lauren Margulieux
Georgia State University

Briana Baker Morrison
University of Nebraska at Omaha

Baker Franke
Code.org, Chicago, IL

Harivololona Ramilison
University of Nebraska at Omaha

Effect of Implementing Subgoals in Code.org's Intro to Programming Unit in Computer Science Principles

Lauren E. Margulieux
Department of Learning Sciences, Georgia State University, Atlanta, GA, USA, lmargulieux@gsu.edu

Briana B. Morrison
Computer Science Department, University of Nebraska Omaha, Omaha, NE, USA, bbmorrison@unomaha.edu

Baker Franke
Code.org, Chicago, IL, USA, baker@code.org

Harivololona Ramilison
Computer Science Department, University of Nebraska Omaha, Omaha, NE, USA, hramilison@unomaha.edu

ABSTRACT

The subgoal learning framework has improved performance for novice programmers in higher education, but it has only started to be applied and studied in K-12 (primary/secondary). Programming education in K-12 is growing, and many international initiatives are attempting to increase participation, including curricular initiatives like Computer Science Principles and non-profit organizations like Code.org. Given that subgoal learning is designed to help students with no prior knowledge, we designed and implemented subgoals in the introduction to programming unit in Code.org's Computer Science Principles course. The redesigned unit includes subgoal-oriented instruction and subgoal-themed pre-written comments that students could add to their programming activities. To evaluate efficacy, we compared behaviors and performance of students who received the redesigned subgoal unit to those receiving the original unit. We found that students who learned with subgoals performed better on problem-solving questions but not knowledge-based questions and wrote more in open-ended response questions, including a practice Performance Task for the AP exam. Moreover, at least a third of subgoal students continued to use the subgoal comments after the subgoal-oriented instruction had been faded, suggesting that they found them useful. Survey data from the teachers suggested that students who struggled with the concepts found the subgoals most useful. Implications for future designs are discussed.

CCS CONCEPTS

• Social and professional topics~Computer science education • Social and professional topics~K-12 education

KEYWORDS

Subgoal learning, instructional design, Computer Science Principles, K-12, Code.org

1 Introduction

The Advanced Placement (AP) Computer Science Principles (CSP) course was designed to be an entry-level computer science course for high/secondary school students [10]. One of the major goals was to create a course that appealed more broadly than AP Computer Science A to increase the number and range of students who engage in computing courses [8]. To engage new students in computing courses, the number of teachers who taught computing courses also had to increase. Many teachers, however, do not have deep computing

content knowledge or experience teaching computing [20]. To provide resources that enable novice computing teachers to offer CSP, Code.org developed 1) a curriculum with a web-based delivery platform that teachers could use with their students and 2) a professional development program that prepared them to use these resources.

Code.org's resources are widely used. In the 2017-18 school year, approximately 45,000 students and 2,800 teachers were active in Code.org's CSP course in the US. Approximately one-third of US students who sit for the AP CSP exam use all or part of Code.org's CSP curriculum. Code.org's professional development program for CSP trains approximately 1500 teachers each year, most of whom are new to teaching computer science. The median teacher has 0-1 years of computer science teaching experience prior to entering the program. Teachers' general lack of experience (and presumably CS content knowledge) is particularly germane for the subgoal learning intervention that was used for this study.

We chose to employ subgoal learning in Code.org's web-based CSP course for the unit on programming to provide consistent, expertise-based explanations of problem-solving procedures for students whose teachers might have little formal computing education. The expertise-based explanations are provided in the form of subgoal labels—short instructional explanations that focus on the tasks achieved within a problem-solving procedure. We developed subgoal labels using a task analysis protocol and then implemented them within the programming unit of Code.org's CSP course. To explore the efficacy of the intervention, we compared the work of students going through the original content of the unit in parallel with those going through the unit with the subgoals intervention embedded. We compared groups using data from open-ended response and multiple choice assessment questions that are identical in both versions of the unit. To augment these learning outcomes data, we collected data about the learning process by examining how the subgoal students engaged with the subgoal comment blocks in AppLab (the programming environment within Code.org's CSP course). We also surveyed the teachers in the intervention group to provide addition context. Our research questions were:
RQ1: How do students use subgoal comment blocks throughout the Intro to Programming unit?
RQ2: What effect does the subgoal instructional material have on responses to assessment questions?
RQ3: How do teachers perceive the usefulness of the subgoal-oriented materials?

2 Background Literature

The subgoal learning framework was designed to help students who are learning problem-solving procedures by bridging the expert-novice gap [6]. The expert-novice gap causes experts to have difficulty verbalizing problem-solving procedures at a level that is understandable to novices because they have automatized much of the knowledge [3]. Knowledge that has been automatized seems like common knowledge or intuition. For many people, trying to describe how to tie a shoe illustrates the difficulty that experts can have verbalizing automatized knowledge about a procedure. The subgoal learning framework bridges the expert-novice gap by identifying low-level procedural knowledge and making it explicit in instructional materials about problem-solving procedures, such as worked examples [6]. Making the knowledge explicit in the instructional materials offloads this difficult task from instructors and ensures that all students receive the same level of instructional support.

Subgoal labeling is a specific technique used to promote subgoal learning. It has been used to help learners recognize the fundamental structure of the problem-solving procedure [4–6]. Subgoal labels are function-based instructional phrases that explain to the learner the purpose of that step, or subgoal, in the problem-solving process. Studies [1,2,4–6,13, 14] have consistently found that subgoal-oriented instructions improved problem-solving performance across a variety of STEM domains, such as programming (e.g.,[15]) and statistics (e.g.,[6]).

Giving subgoal labels in worked examples improves learner performance while solving novel problems without increasing the amount of time learners spend studying instructions or working on problems [15]. Subgoal learning was first applied to programming education in the context of an experimental laboratory with psychology undergrads as participants [15]. Due to this context, the programming procedure being taught had to be accessible to absolute novices. Thus, participants were taught to create apps in Android App Inventor. In this highly controlled environment, subgoal labeled worked examples were found to improve problem-solving performance by 8%. From that experiment, research has focused on testing subgoal labeled worked examples in more authentic programming education environments, including online learning with K-12 teachers [14], a game-based K-3 setting [11], and in open educational resources that crowdsource subgoal labels [12]. Our research applies subgoal learning to an introductory programming course, specifically to students who were learning to solve problems using while loops.

Decker, Margulieux, and Morrison have completed several studies implementing subgoals in text-based programming [9,16–19]. They have found that using subgoal labels to teach `while` loops produced results similar to those achieved in other disciplines. Their first study [19] found that students who learned with subgoal labels (either given or self-created) performed better on the code writing assessments than those who learned without subgoal labels. In a follow up paper [18], the authors examined the performance of students on a specific type of problem assessment, Parsons problems, after having learned loop problem solving in one of the treatment groups (with no subgoal labels, with given subgoal labels, or generating their own subgoal labels). They found that students who were given subgoals performed statistically significantly better than those who had no subgoals or who generated their own subgoals, regardless of transfer condition. In [17] the authors replicated their initial study [19] with a third population of students. The results supported the findings from the previous studies: participants who learn by generating subgoal labels performed the best.

Recent work on subgoal learning has applied the framework to programming education [20], especially in web-based learning environments [15]. Given the success of these recent applications, employing subgoal learning in Code.org's web-based CSP course for the unit on programming seemed like a natural fit, though the motivation was different. In this case, the motivation was not to bridge the expert-novice gap but instead to provide consistent, expertise-based explanations of problem-solving procedures for students whose teachers might have little formal computing education. The expertise-based explanations are provided in the form of subgoal labels—short instructional explanations that focus on the tasks achieved within a problem-solving procedure. Subgoal labels are expertise-based because they are identified through a task analysis with an expert.

The task analysis protocol was created by Catrambone as a means for uncovering the tacit knowledge of an expert in the problem solving process [7]. The protocol involves an instructional designer working with a subject-matter expert to identify automatized procedural knowledge, verbalize it, and then design a series of subgoal labels that can be used in instructional materials. For a brief summary of the protocol, the instructional designer asks the subject-matter expert to solve problems while explaining why they are doing each step. As the expert solves multiple problems within a class, the designer identifies common rules and decision points. When the expert has difficulty explaining why a step was taken or why a decision was made, the designer asks questions to extract this automated knowledge. Eventually, the designer tries to solve problems using only the common rules and steps identified through the task analysis. When the designer can consistently solve any new problem in a class, the task analysis is complete [7]. A full description of the task analysis protocol can be found in Margulieux et al. [16].

Block-based programming languages offer users menus of pre-formatted lines of codes to eliminate the possibility of syntax errors and limit the search space for potential actions. The task analysis protocol done for block-based programming languages yields higher-order subgoals than those generated for text-based languages. For example, while two of the subgoals for creating an app in block-based Android App Inventor

are “Handle input,” and “Set output” [14], the subgoals for writing a selection statement in text-based Java include “Write if statement with Boolean expression,” “Follow with true bracket including action,” “Follow with else bracket,” etc. [16].

The difference in the level at which the subgoals apply are notably different. In a text-based programming language the subgoals generated have been at the programming construct level: how to design and implement a specific control structure or program component. The block-based subgoals were at the level of problem-solving. In App Inventor, learners who want to use a selection statement simply need to drag out an `if` block from the menu. Deciding to use an `if` block is the major cognitive contribution in App Inventor. However, in Java, the learner has to determine when a selection statement is needed and then write the statement syntactically correct. The subgoals identified for AppLab—Code.org’s block-based JavaScript programming environment—were higher order subgoals, like “Define function” and “Write a loop.” The Code.org curriculum also makes use of turtle graphics which led to subgoal labels like “Move turtle” and “Orient turtle”.

To identify the subgoals for the programming unit of Code.org’s CSP course, the task analysis protocol was completed between authors Margulieux and Morrison. Margulieux has completed this protocol in a variety of STEM fields, such as programming and chemistry, and Morrison has served as subject-matter expert in this protocol for topics in introductory programming, such as writing loops and selection statements. The subgoal labels designed through this process can be split conceptually into AppLab/turtle graphics subgoals and general programming concept subgoals. This distinction was not included in the instructional materials that students used, but we make it here for readers to separate the subgoals that apply only in programming environments like ours and the subgoals that apply more generally to programming procedures.

- AppLab/Turtle Graphics subgoal labels
    - Move turtle
    - Orient turtle
    - Set pen properties
- General programming concept subgoal labels
    - Define function
    - Call function
    - Write a loop

3 Implementation in the Programming Unit

For the 2017-2018 school year, the Intro to Programming unit in Code.org’s CSP course had three introductory lessons (1-3) and six interactive programming lessons (4-9; see Figure 1). Each programming lesson starts (see Figure 2) with learning objectives, vocabulary, and new blocks to use in AppLab. Lessons comprised sequences of small tasks, called a “level”. Each lesson has multiple programming levels (see Figure 3) that include some instructions and ask learners to complete tasks or solve problems in AppLab, like using turtle graphics to draw a square. Programming levels can be configured in AppLab to have a blank workspace, include pre-populated blocks to scaffold the activity, or use pull-through code that the learner builds up while working through a sequence of multiple levels. Lessons also include several assessment levels (see Figure 4) that include open-ended response and multiple choice questions. Each lesson includes approximately 15 total levels.

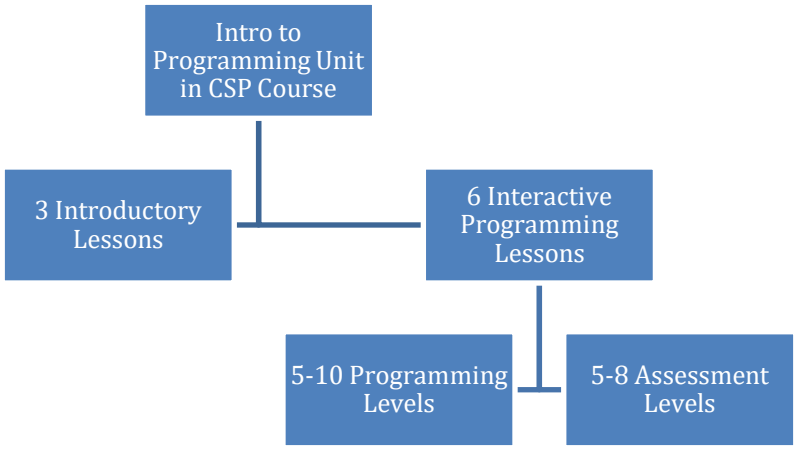

Figure 1. Hierarchy of components that are included in the Intro to Programming Unit of Code.org's CSP course.

To add subgoal labels to the Intro to Programming unit, we redesigned the 6 interactive programming lessons to include subgoals. The initial redesign was completed by author Margulieux, who has eight years of instructional design experience, has taught instructional design, and is the coordinator of a PhD program in instructional design and technology. To design the subgoal integration, she used the Dick and Carey model of instructional design [10] that is based on the fundamental ADDIE (Analysis-Design-Development-Implementation-Evaluation) framework of instructional design. The Dick and Carey model is an iterative model that starts with analyzing the needs of the learners (i.e., the gap between current and desired knowledge). This analysis is used to design the learning objectives, which are then used to create the assessments. Then the instructional strategy and materials are developed to bridge the learning objectives and assessments. Finally, formative and summative evaluation of the instructional system are conducted. Formative feedback and iterations of the redesign were completed by author Morrison, who has 23 years of experience teaching programming; Baker Franke, who led the original design of Code.org's CSP course offered technical and engineering assistance to the researchers for the redesign of the programming unit; and Bryan Cox, who is a former high school CS teacher and is now the Computer Science Specialist at the Georgia Department of Education.

The first level of each lesson is a static student-facing page with a brief lesson summary and important vocabulary. As part of the redesign of each lesson we added to this page the new subgoals that corresponded to the new blocks (see Figure 2). To the programming levels, we added a "Goals" category within the AppLab code toolbox which contained subgoal labels as comment blocks (see gray // blocks in Figure 3). The subgoal blocks could be dragged into the workspace like any other block, and they could also be edited. To the instructions (top right of Figure 3), we added which subgoals were needed to complete the puzzle (e.g., in lesson 5, we added to a level's instructions, "To solve this puzzle, you'll use three types of subgoals. You'll *define the function*, which will include *calling 2 different functions* and *moving the turtle*. Then you'll *call the function* that you defined"). We implemented a faded scaffolding approach by adding these additional instructions only to the first four out of six lessons. In the first two lessons we prepopulated the AppLab workspace with some subgoal blocks, to act as an outline for the solution, as an example for students and to narrow the problem-solving space. Therefore, the first third of lessons included both explanations of the subgoals in the problems and prompts to use subgoal blocks ("Subgoal Prompted"), and the second third of lessons included explanations but not prompts ("Subgoal Explained"). For the final third of the lessons the scaffolding completely faded. The subgoal blocks were still available in the toolbox, but the instructions did not specifically discuss subgoals nor did the workspace have prepopulated subgoal blocks ("Subgoal None").

Figure 2. First level of lesson 4 (first programming lesson), including learning objectives, vocabulary, new subgoals and blocks. Subgoals are near the bottom of the screen.

The assessment levels for the subgoal intervention were identical to the original content, however, we added eight assessment items to both the original unit, which the control group received and the subgoal intervention unit. The new assessments were specifically designed to examine differences in thought processes between learners who had subgoal labels and those who did not. One of these new items is in Figure 4, and more are discussed in the presentation of the results.

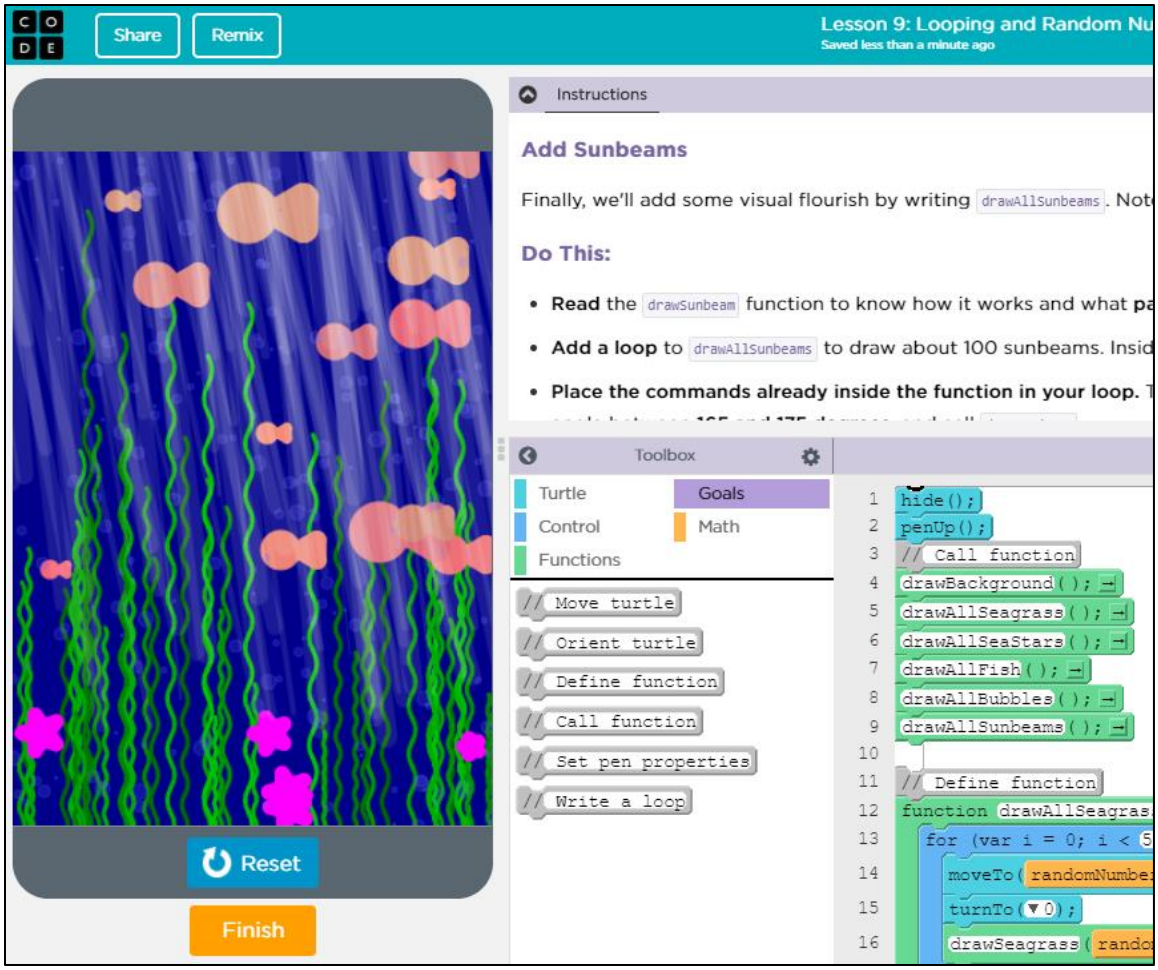

Figure 3. Programming level in lesson 9 (final lesson), including all subgoal comment blocks in the toolbox and block code in the workspace at the bottom right.

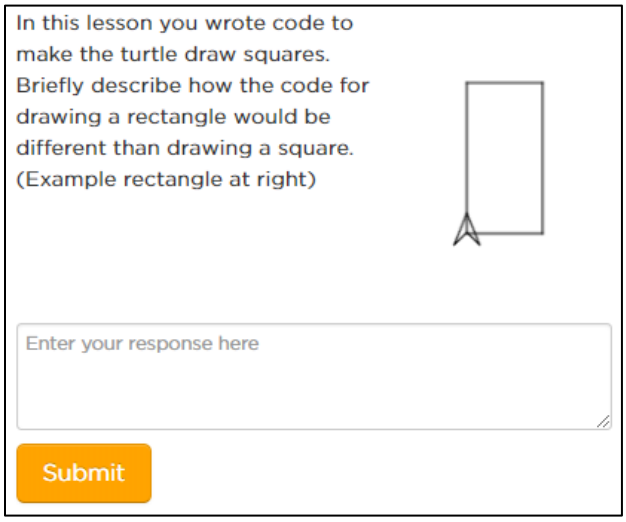

Figure 4. Open-ended response assessment level.

4 Method

To address these research questions, we compared data from students who worked in the redesigned subgoal-oriented unit and those who worked in the original unit during the 2017-18 school year. To make the learning experience as authentic as possible, the researchers did not impose any restrictions on how students or their teachers engaged with the unit. To recruit participants, Code.org emailed all active teachers of the course to ask them if they would like to participate in a study. The recruitment email explained the subgoal learning framework and that they should use Code.org's materials as they regularly would. The participants never directly interacted with the researchers, though they were given their contact information in case they had questions. If participants opted into the study, author Franke added them into a cohort grouping on the Code.org platform that transparently swapped out the original unit for the redesigned subgoal unit. The control group were the remaining users of Code.org's CSP course.

Due to the recruitment procedures, the researchers could be given access only to data that followed Code.org's privacy policy and user agreement. Therefore, researchers never received identifying information nor raw data that were created by students. For example, the researchers could not receive the raw text of students' open-ended response answers or student-written code because they might include personally identifying information. Instead, the researchers were allowed to run queries for keywords against the data sets of the open-ended response assessments. Anyone can access the subgoal-based unit or the data used for analyses, but we are not permitted to share them publicly. Interested parties can contact author Franke from Code.org. In addition to data from Code.org's platform, the researchers asked the teachers who had opted into the study to complete a survey after finishing the programming unit. Due to IRB consent procedures, this data cannot be shared outside of the research team.

4.1 Data Collection Sources

Within Code.org's privacy policy, researchers had access to three types of data sources from the platform. The first type was the multiple choice assessment levels (i.e., product data) that were built into the lessons. Because responses to the multiple choice questions do not include information created by the student, we could collect student answers to the questions. The multiple choice questions included in the lessons were designed to provide formative feedback on the students' progress to both the students and teachers. Thus, these questions have a different goal than multiple choice questions that would be designed to measure student performance in an experiment. Instruments used in experiments aim for a mean score of 50% to avoid ceiling effects, while instruments used for formative feedback aim to measure a base competency, ideally with most students getting questions right on the first try. Therefore, the effect size between groups on these questions were expected to be small due to ceiling effects.

The other two types of data collected were (a) students' answers to open-ended response assessment questions (e.g., Figure 4, product data), and (b) student-written code for AppLab programming levels (e.g., Figure 3, process data). Due to the possibility of these data containing identifying information we did not collect the raw data, but we designed queries for Code.org to execute on our behalf against the data and return the results de-identified. For both the open-ended response text and AppLab code we queried for instances of whole or partial use of the subgoal comment names and blocks. These are detailed more in the Results section.

In addition to data provided from Code.org, we asked teachers who opted into the study to complete a survey. The survey included four fixed-response questions about their students' perception of the subgoal-oriented instruction and how this year's students' performance compares to last year's, if they taught CSP the previous year. The scale for these questions was a classic Likert scale from "1-Strongly disagree" to "5-Strongly agree." Because we were asking the teachers about their students' perceptions and performance, we included an

"Unsure" option to avoid a forced, inappropriate choice. We also asked teachers an open-ended question: "Please tell us anything that you'd like to share about the subgoal-oriented programming unit."

4.2 Data Cleaning Methods

The data supplied to the researchers from Code.org came in multiple hierarchical database tables. In order to run statistical tests, the data had to be "flattened" into a single table. Data received from Code.org from the queries were stored in separate tables according to the type of assessments (i.e., multiple choice, open-ended response, or AppLab). Automated processes were developed to transform the hierarchical data into a single table. Data transformation consisted of converting the structure of the multiple tables into a unified structure that would link multiple pieces of data from the same participant. This procedure extracted all the programming or assessment levels and related responses provided by a single participant into separate columns. At the end of the data transformation process, all participation responses were in a single large, table with each row representing a participant and the columns containing the assessment responses. This table then went through a data removal process that eliminated incomplete data by applying certain rules.

Given that anyone with internet access can use Code.org's resources, we examined the data to determine typical usage. We found a large gap between users who engaged with most of the content in the unit (i.e., likely assigned to complete the unit by their teacher in a formal course) and those who engaged with only a few elements (i.e., likely given the unit as an optional resource by their teacher or someone looking through the unit). Based on this completion gap, we created rules to eliminate data from occasional users. We removed students who did not answer at least one multiple choice question and one open-ended response assessment. This initial data removal yielded a subgoal group of $n = 2027$ and a control group of $n = 10206$. Based on the large sample size, we then created more selective inclusion rules. In the second round of data removal, we eliminated students who answered fewer than five multiple-choice questions, four open-ended response levels, and six AppLab levels, leaving us data for students who completed at least 70% of the unit. These rules narrowed the subgoal group from $n = 2027$ to $n = 1413$, a 30% decrease, and the control group from $n = 10206$ to $n = 6740$, a 34% decrease.

4.3 Threats to Validity

Because teachers elected to participate in the subgoal-oriented unit, the intervention and controls groups were not randomly assigned, which introduces sampling bias. Due to the sheer size of the sample, it is unlikely that sampling bias would be a major source of systematic variance between groups because there is a large amount of variance within groups. The authors evaluated the groups for potential bias during analysis. Details are discussed in the Results section, but there does not seem to be a substantial sampling bias based on demographics, teacher experience, or variance within measurements due to elective assignment to the intervention group.

Our goal in the data elimination process was to focus the analysis on students who completed all or most of the unit, likely representing students in a formal classroom setting. Code.org's CSP curricula is unlike an online course for informal learning, like a Massive Open Online Course (MOOC). For Code.org's CSP online resources, the biggest predictor of completion was likely whether students were assigned to complete items. Because the sample sizes are so large and because the attrition between groups is about equal, we do not expect attrition to bias the data in favor of one group. In support of this argument, we also compared the available demographic data for both groups. We had access to basic demographic data (gender, race, and whether the teacher had taken Code.org's Professional Development (PD) or was in their first year of teaching CSP). While much of these data were missing for our final samples, the data available shows that both groups are representative of the national population of CSP students. The entire population of students who use Code.org's CSP course is representative of the national population of CSP students, which is around 30% girls and 40% underrepresented minorities. The criteria we used also ensures that we have more data points about each student in the analysis than if we included everyone who participated in some part of the unit.

5 Results and Discussion

This section is organized by the data sources used in this study. The first section covering the AppLab data addresses the first research question about how the subgoal group engaged with the subgoal blocks. The next two sections, multiple choice and open-ended response assessments, address the second research question to compare answers between the subgoal and control groups. The last section reports on the teacher survey data and addresses the third research question about teachers' perceived usefulness of the subgoal intervention.

5.1 Subgoal Comments Use in AppLab (Research Question 1)

Our first research question asks how students use subgoal comment blocks in the unit, which will be addressed in this section. To explore how students engaged with the subgoal label comment blocks in the AppLab programming levels throughout the lessons, we queried the student code to produce a count of the number of occurrences of each type of subgoal block present in the code. Due to Code.org's privacy policy, we were not able to analyze the complete code that students created, but Code.org does use basic tests against program execution logs to ensure that students' programs meet critical criteria before they are marked as complete. Because we examined the number of subgoal blocks used in programming levels, these data are only available for the subgoal group because the control group did not have the subgoal blocks. For this analysis, we used data from participants who completed all of the programming levels, regardless of whether they completed all of the assessments. Therefore, the *n*, 2026, is higher than that for the assessment data.

Because the data are for only one group, only descriptive statistics were used (see Tables 1-5). The mean and standard deviation (SD) refer to the number of subgoal blocks used per student, including students who did not use any subgoal blocks. The data are generally split by how the students in the intervention were encouraged to use the subgoal blocks. For the first two lessons, the subgoals needed for the tasks were explained to the student, and the instructions asked students to use the subgoal blocks. We use "Subgoal Prompted" to describe these lessons. For the middle two lessons, the subgoals needed for the task were explained, but the instructions did not prompt students to use the subgoal blocks. We use "Subgoal Explained" to describe these lessons. For the final two lessons, the subgoals needed to solve the puzzle were neither explained nor prompted. We use "Subgoal None" to describe these lessons.

For the Subgoal Prompted lessons, 88.5% of students used the subgoal blocks at least once. This means that 11.5% of students did not use the subgoal blocks at all, even though in some of the programming levels, the subgoal blocks were already pre-populated in the workspace to guide problem solving. Most students added additional subgoal blocks, and 22% of students added 25-40 more blocks, meaning that they were frequently using the blocks.

Table 1 lists the number of subgoal blocks in students' code for these levels. For example, in the Subgoal Prompted lessons, the mean number of subgoal blocks used was 24.5 per lesson, found in the left column. The middle column describes the number of times students used blocks. The distribution of each level was manually evaluated for trends to develop the bins that are reported. These trends are reported as the approximate number of students in parentheses in the middle column. For example, the number of students who used 1-6 blocks was about 30 for each (i.e., about 30 students used the blocks once, about 30 students used the blocks twice, etc.). The right column lists the total percentage of students who are included in the bin. For example, 10% of students used the blocks 1-6 times. In Table 1, there is a spike for seven subgoal blocks used because that is the number of pre-populated subgoal blocks in the workspace. Therefore, if students used fewer than seven blocks (10% + 12% who used 0 blocks), they removed blocks; if they used seven blocks (17%), they did not add any blocks; and if they used more than seven blocks (61%), they added some blocks. Most students added additional subgoal blocks, and 22% of students added 25-40 more blocks, meaning that they were frequently using the blocks.

Table 1. Subgoal block usage comparing instructions that prompted students to use blocks, explained which subgoals were used but did not prompt use, or did neither. * indicates the number of pre-populated blocks in the workspace.

Instruction	# times subgoal used (approx. # of students)	% of students
Subgoal prompted (M = 24.5, SD = 28.7)	0 times (229 students)	12%
	1-6 times (~30 students)	10%
	7 times (342 students)*	17%
	8-11 times (~50 students)	8%
	12-31 times (~18 students)	17%
	32-47 times (~28 students)	22%
	48-184 times (<10 students)	14%
Subgoal explained (M = 3.9, SD = 6.9)	0 times (1224 students)	61%
	1-3 times (~68 students)	10%
	4-13 times (~30 students)	16%
	14-50 times (<20 students)	13%
Subgoal none (M = 5.9, SD = 15.6)	0 times (896 students)	51%
	1-2 times (~150 students)	15%
	3-6 times (~65 students)	14%
	7-23 times (~26 students)	15%
	24-195 times (<10 students)	5%

After the two Subgoal Prompted lessons, no subgoal blocks were pre-populated in the workspace for the remaining four lessons. For the Subgoal Explained lessons, 38.5% of students continued to use subgoal comments at least once. The instructions for these lessons were fairly well-defined, and when students faced more open-ended problems in the Subgoal None lessons, they were more likely to use the subgoal blocks. For the Subgoal None lessons, 49.3% of students used subgoal comments.

To further explore how students continued to use subgoal blocks after they were prompted to do so, we examined the use of subgoal blocks that were introduced after the Subgoal Prompted lessons. "Set pen properties" was introduced in the Subgoal Explained lessons, and "Write a loop" was introduced in the Subgoal None lessons. **Even though the instructions did not prompt their use, both of these subgoal comments were used by many students**, often multiple times. These results suggest that students found the subgoal comments useful because they continued to use them, and started using new ones, after they were no longer required to (see Table 2).

Table 2. Subgoal block usage for subgoals introduced in later lessons.

Instruction & "Subgoal"	# times subgoal used (approx. # of students)	% of students
Subgoal Explained "Set pen properties" (M = 2.6, SD = 2.9)	0 times (1256 students)	64%
	1 time (157 students)	8%
	2-8 times (~75 students)	25%
	9-21 times (<25 students)	3%
Subgoal None "Set pen properties" (M = 2.3, SD = 4.9)	0 times (1476 students)	77%
	1-4 times (~60 students)	15%
	5-9 times (~18 students)	5%
	10-68 times (<10 students)	3%

Subgoal None "Write a loop" (M = 1.0, SD = 2.0)	0 times (1469 students)	77%
	1-2 times (~110 students)	7%
	3-6 times (~30 students)	11%
	7-19 times (<10 students)	5%

Tables 3-5 show the progression of how individual labels were used among the Subgoal Prompted, Explained, and None lessons. These data suggest that the number of students who used many blocks reduced drastically in later lessons, but many students continued to use blocks occasionally in later lessons.

Table 3. Subgoal block usage for each subgoal during early lessons when students were prompted to use blocks. * indicates the number of pre-populated blocks in the workspace.

Subgoal Prompted	# times subgoal used (approx. # of students)	% of students
Move turtle (M = 11.4, SD = 13.6)	0 times (211 students)	14%
	1-3 times (~50 students)	8%
	4 times (527 students)*	26%
	5-16 times (~30 students)	18%
	17-20 times (~100 students)	19%
	21-100 times (<18 students)	15%
Orient turtle (M = 10.3, SD = 12.8)	0 times (322 students)	17%
	1-2 times (~45 students)	4%
	3 times (488 students)*	24%
	4-14 times (~36 students)	19%
	15-19 times (~92 students)	21%
	20-84 times (<20 students)	15%
Define function (M = 1.4, SD = 2.6)	0 times (1176 students)	62%
	1-2 times (~200 students)	19%
	3-8 times (~50 students)	14%
	9-16 times (10 students)	5%
Call function (M = 1.5, SD = 4.1)	0 times (1344 students)	68%
	1-2 times (~200 students)	20%
	3-8 times (~25 students)	7%
	9-45 times (<10 students)	5%

Table 4. Subgoal block usage for each subgoal during middle lessons when subgoals were explained but their use was not prompted.

Subgoal Explained	# times subgoal used (approx. # of students)	% of students
Move turtle (M = 1.4, SD = 2.6)	0 times (1413 students)	70%
	1-5 times (~50 students)	21%
	6-22 times (<14 students)	9%
Orient turtle (M = 1.7, SD = 2.1)	0 times (1487 students)	79%
	1 time (113 students)	6%
	2-4 times (~75 students)	11%
	5-10 times (<20 students)	4%

Define function (M = 0.4, SD = 0.8)	0 times (1746 students)	89%
	1 time (157 students)	8%
	2 times (~38 students)	2%
	3-9 times (<10 students)	1%
Call function (M = 0.4, SD = 0.8)	0 times (1739 students)	89%
	1 time (156 students)	8%
	2-9 times (<12 students)	3%

Table 5. Subgoal block usage for each subgoal during later lessons when subgoals were neither explained nor prompted.

Subgoal None	# times subgoal used (approx. # of students)	% of students
Move turtle (M = 1.2, SD = 4.9)	0 times (1649 students)	83%
	1-4 times (~50 students)	10%
	5-47 times (<15 students)	7%
Orient turtle (M = 1.0, SD = 4.4)	0 times (1665 students)	84%
	1-2 times (~65 students)	7%
	3-8 times (~15 students)	6%
	9-65 times (<10 students)	3%
Define function (M = 1.0, SD = 3.3)	0 times (1598 students)	80%
	1 time (121 students)	6%
	2-8 times (~30 students)	11%
	9-45 times (<10 students)	3%
Call function (M = 1.0, SD = 3.4)	0 times (1512 students)	73%
	1-2 times (~150 students)	15%
	3-7 times (~40 students)	9%
	8-25 times (<10 students)	3%

Much of the later use of subgoals was in the final lesson that had more complex programs (e.g., the seascape program in Figure 3) and an open-ended play level. This usage aligns with the subgoal comment blocks' purpose, to decompose large problem-solving spaces into smaller pieces that students can focus on yet still be aware of how smaller pieces contribute to the whole program. The practice of decomposition is critical in programming, and it is possible that the subgoal comments helped formalize students' development of that practice. This conclusion is not supported with the AppLab data alone, it is further backed up by the surveys that teachers completed (see section 4.4).

5.2 Multiple Choice Assessments (Research Question 2)

Our second research question asks what effect the subgoal instructional material had on assessments, which will be addressed in the next two sections. Throughout the six interactive lessons of the programming unit, students answered 15 multiple choice questions. Students who did not answer one of the questions were eliminated from analysis, which eliminated 1% of the participants from both the subgoal group (final n = 1384) and control group (final n = 6624). Because the percentage of participants excluded from analysis is small, attrition was not considered a significant source of error.

The questions were separated into three types. There were nine knowledge questions (e.g., "Which of the following is NOT a true statement about functions?" or "What is an API?"), five perception questions (e.g., "I like computer science," or "I have the ability to learn computer science,"), and two similarity questions (i.e., "Which of the following blocks is least similar to the others?"). The two similarity questions were added

by researchers to *both* the original unit (being used as a control) *and* subgoal units to determine whether subgoal learning encouraged students to recognize functional similarities among different code blocks. In Figure 5, choices A, C, and D all deal with moving or drawing with the turtle, and choice B deals with orienting the turtle. Choice C is the distractor item of the list because it both moves and changes the orientation of the turtle. This explanation is not intended to say that B is the only correct answer, but selecting B would indicate that the students are basing similarity on these functional features, which are the features that the subgoal labels were intended to highlight.

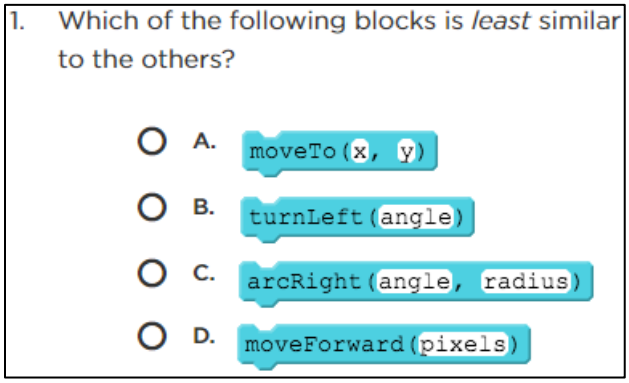

Figure 5. Multiple choice question for functional similarity.

Given this intent, the similarity data were coded for whether the student selected the functionally different block, and the coded data were combined into a single score. Levene's test (mean) for homogeneity was non-significant despite the large difference in sample sizes between the groups, so it was considered appropriate to use the entire sample. In addition, due to the large sample size, the central limit theorem suggests that the data are normally distributed, which we confirmed with visual inspection of distributions. Students who learned with subgoal labels were more likely (78% of participants answered both questions based on functionality) to make similarity judgments based on functionality than students in the control group (67%), t (8006) = 39.4, $p < .001$, $d = .22$. This finding is meaningful because categorizing blocks by functionality can help students to transfer their knowledge to novel problem solving [7]. In addition, two-thirds of control group students also judged similarity on functionality, suggesting that it is a useful categorization scheme. Therefore, the finding that the subgoal group was more likely to use this categorization suggests that the subgoal labels helped them to mentally organize blocks in a useful way.

The usefulness of subgoal labels is further supported by examining student responses to the knowledge-based multiple choice questions. The responses to knowledge questions were coded for whether the answer was correct or not and summed for a total correct score. Levene's test (mean) was again non-significant, so it was considered appropriate to use the entire sample. Overall, the subgoal group performed better than the control group, t (8006) = 25.5, $p = .001$, $d = .18$, however an interesting distinction between questions provides a more nuanced explanation of the difference.

The subgoal group performed better than the control group on questions that asked about the problem-solving procedure (e.g., "Which of the following statements about writing functions and top-down design is NOT true?" or "Which line of code should be removed to make the program do what it's supposed to?"). In contrast, the two groups performed equally on questions about declarative knowledge (e.g., "Which of the following is NOT a valid use of randomNumber?" or "What is a function parameter?"). If only the four procedure-focused questions are included in the analysis, then the effect size nearly doubles, t (8006) = 11.1, $p < .001$, $d = .34$. This distinction between procedural and declarative knowledge aligns with the subgoal learning framework, which is intended to help students learn problem-solving procedures. These data indicate that,

though the subgoal labels improved performance on procedural multiple choice questions, they did not improve performance on declarative multiple choice questions.

To explore subgoal labels' effect on the experience of learning, we examined the five perception multiple choice questions (listed below), which used a 5-point Likert scale from Strongly Disagree to Strongly Agree. Overall, the subgoal group more strongly agreed with the statements than the control group, F (1, 8006) = 3.69, p = .005, η^2 = .01, but again an interesting distinction among questions arose.

The subgoal group agreed more strongly than the control group with questions about their attitudes about and self-efficacy towards computer science (i.e., "I like computer science," "I have the ability to learn computer science," and "I want to take more computer science classes in the future."). In contrast, both groups responded similarly for questions about their classroom environment (i.e., "I like this computer science class," and "I feel comfortable in this computer science class."). If only the three general questions are included in the analysis, then the effect size increases again, F (1, 8006) = 5.02, p < .001, η^2 = .02. Therefore, the subgoal labels have a positive effect on students' general attitudes toward computer science, but they have no effect on their attitudes toward this specific class.

Based on the authors' experience with classrooms that use Code.org's CSP course and reports from teachers, these findings are likely due to students' feelings about their local classroom environment compared to their global feelings about computer science. In general, the students in CSP like their computer science class and their teacher, and these attitudes would be difficult to change with a subtle instructional manipulation. However, the subgoal manipulation might help students feel that they understand the problem-solving procedure, improving their sense of learning and self-efficacy. These findings were unexpected, so more research would be needed to explore the effect of instructional manipulations on local versus global perceptions of computer science.

In general, the effect size between groups for the multiple choice questions was small, d = .18. As explained in section 3.1 on data collection sources, we expected a small effect due to questions' design to provide formative feedback rather than measure experimental performance. In addition, this effect size is reasonable for multiple choice questions because they inherently include significant error variance. Most of the multiple choice questions had five choices; therefore, if students randomly selected answers, about 20% of them should be correct due to error, which decreases effect size. Furthermore, we had no experimental control over the classroom environment whiles students used the materials, meaning various sources of error could also reduce the effect size. Because some items had a difference between groups on means and some items had almost exactly the same means between groups, the items with a difference, even though the effect size is small, represent a consistent, meaningful difference due to subgoal labels.

5.3 Open-ended Response Assessments (Research Question 2)

To continue to address our research question about the effect of subgoal instructional materials, we pair multiple-choice assessments with open-ended assessments. Throughout the six interactive lessons of the programming unit, students answered 18 open-ended response questions. Due to Code.org's privacy policy, the researchers could not access original content created by the students. Instead, the researchers were able to design and run queries that produced data about characteristics and features of the responses, including total number of characters and counts of keywords. The keywords selected for analysis were words from the subgoal labels—move, orient, define, call, function, pen properties, and loop—to examine whether students who received subgoal labels were more likely to use those words when answering the questions, indicating that the labels affected their thought processes. Contrary to expectations, however, there were few differences between the groups in how often they used the keywords. For the AppLab/Turtle Graphic subgoals (move, orient, and pen properties), students in either group did not use the words (i.e., each was found fewer than 20 times). For the programming concept subgoals (define, call, function, and loop), students in both groups used

them at the same rate (i.e., most students used them once or twice in questions that corresponded to each concept). We forgo a formal statistical evaluation to preserve space and instead focus on differences between groups in the total number of characters written.

In addition to previous data cleaning, participants who had 0-4 characters in any of the 18 open-ended response questions were excluded from analysis. Based on frequency counts of number of characters for each question, there were several students who had 0-4 characters, indicating a superficial level of engagement with the question, and a steep drop-off for 5+ characters, which is how this cut-off was chosen. From the subgoal group, 3% of participants were excluded from analysis for a final n = 1365, and from the control group, 2% of participants were excluded for a final n = 6573. Because the percentage of participants excluded from analysis is small, attrition was not considered a significant source of error.

The total number of characters for each of the 18 responses had a mean between 100 and 450, with most between 150 and 250, and different length responses were appropriate for different questions. To normalize the number of characters across different questions, z-scores were calculated. Creating z-scores allows for the responses to be analyzed with a repeated-measures ANOVA. Z-scores exceeding four (i.e., four standard deviations above or below the mean) would have been removed as outliers, but none were found. Repeated-measures ANOVA is ideal for this kind of analysis because it connects different responses from the same participant together to disregard much of the personal variance among participants that is not due to the intervention.

Despite the large difference in sample sizes, Levene's test (mean) of homogeneity was not significant for 16 of 18 open-ended response questions, so ANOVA was considered appropriate. Researchers decided to keep the other two responses in the full analysis rather than analyze them separately to better account for individual variance, but responses for these two questions will be discussed in detail. Overall, the subgoal group wrote more characters per question than the control group, F (1, 7936) = 61.7, $p < .001$, $\eta^2 = .08$.
Calculating an effect size for difference between means, like Cohen's d or f, is not possible for repeated-measures, but there were two levels of effect based on inspection of the results. For 8 of the questions, the difference between groups was a smaller effect of about 15-20 characters—likely 4-5 words. These questions tended to be about relating computing to everyday experiences, such as:

- "Describe the features of a programming language that make it different from the language you typically use in everyday life. Explain why a programming language must be created in this way,"
- "This lesson introduced the notion of 'efficiency' in programming, and that it might mean different things at different times. Think of an example outside of computer science in which you have heard the term 'efficiency' and compare it to the ways we talked about efficiency in programming."

For another eight of the questions, the difference between groups was a larger effect of about 30-40 characters—likely 8-10 words. These questions tended to be about programming knowledge or problem solving, such as:

- "It is said that functions with parameters generalize the behavior of a more specific command and allow programmers to use functions instead of duplicated code. Explain what this means to you using the difference between turnLeft() and turnLeft(angle) as an example,"
- "When breaking a problem down, you often encounter elements that you want to use repeatedly in your code. Sometimes it's appropriate to write a new function; at other times it's appropriate to write a loop. There is no hard-and-fast rule as to which is better, but what do you think? What kinds of circumstances would lead you to writing a function versus using a loop?"

Similar to the multiple choice data, the subgoal intervention had a larger effect on questions related to problem-solving procedures than to other topics. Unlike in the multiple choice data, the subgoal intervention still had a small effect on the other types of questions. Because open-ended response questions are open-ended, they are more sensitive than fixed-response questions to differences between groups. These findings paired with better performance on the multiple choice questions suggests that the subgoal group had higher

fluency when answering the open-ended response questions. This possible explanation is particularly poignant for responses to the two open-ended response questions that violated Levene's test.

The two questions that violated Levene's fell outside of the smaller and larger effect size groupings. For one question, the subgoal group wrote 102 characters more (M = 451, SD = 72) than the control group (M = 348, SD = 53) in response to a question presented in a lesson as practice response to a prompt for the AP Exam Performance Task,

- "Try to write a response to this AP Prompt thinking about either how you developed the idea for the snowflake drawing program, or how you resolved to make the 3x3 grid program. You might have to use a little bit of imagination to assume that it's part of a larger program you created yourself. The point is to practice writing about your development process."

The subgoal group on average wrote about 100 words and 30% more than the control group, which is likely more appropriate for this kind of complex question than a shorter answer. Though we cannot qualitatively analyze the students' responses, we consider their performance better than the control group on this question, suggesting that subgoals help students to more fluently express their knowledge.

The subgoal group did not always write more than the control group. For the other Levene-violating question, the two groups had only a 3-character difference between means in response to the question,

- "Consider the figure below. Use top-down thinking to design a solution to the problem. In the space provided write a list of just the names of the functions that you would write in a program that draws this figure."

Given the constraints of the question, much variation would not be expected. Therefore, this exception to the pattern of results suggests that the subgoal group is not more verbose when they do not need to be. In the other questions, the subgoal group tended to write more thorough answers to open-ended response questions than the control group.

5.4 Teacher Perception Survey (Research Question 3)

Our last research question asked how teachers perceived the usefulness of subgoal instructional materials, which will be addressed in this section. To gain more insight into the perceptions and uses of the subgoal intervention, we asked teachers who had opted into the study to complete a survey after they finished the programming unit. In total, 139 teachers completed the survey. Characteristics of their courses are described in Table 6. Teachers also reported on their students' demographic characteristics. The average number of students per teacher (not per class) was 32.65 but varied greatly, std. dev. = 29, and ranged from 3 to 100. Most students were in 12th grade (39%) or 11th grade (31%) with a smaller number in 10th (23%) or 9th (6%), and less than 1% in 8th grade or unknown. The gender composition was 71% boys, 28% girls, and less than 1% transgender, non-binary, agender, or unknown. Just over half of the students were White or Caucasian (53%) with smaller percentages of students who were Hispanic or Latinx (15%), Asian (13%), or Black or African American (11%), and a small percentage of students who had multiple ethnicities (3%), another ethnicity (1%), or whose ethnic background was unknown to the teacher (4%).

Table 6. Course characteristics of teachers who completed the survey.

Course Characteristics	Most common response (% of respondents)	Mean response
How often does your CSP class meet per week?	5 times per week (55%)	4.56 times per week
How many weeks long is your CSP class?	36 weeks (57%)	35.1 weeks
How many contact minutes per week do you have?	225 minutes (17%) or 250 minutes (16%)	238.6 minutes

About what percentage of Code.org's CSP materials do you use in your class?	>90% (68%)	4.73 (4=75-89%, 5= >90%)

Teachers were asked, "Please tell us anything that you'd like to share about the subgoal-oriented programming unit." Half of the respondents, 71 teachers, responded to this question. We analyzed these qualitative responses with content analysis to identify themes. We did not approach the analysis with an a priori coding scheme. Instead, we looked for common features that emerged from the responses and found six themes.

Multiple teachers commented that the subgoal-oriented materials seemed to particularly help their students who had less prior knowledge or struggled more with the concepts.

> *Students were resistant to use sub-goals initially, however, once they realized it was important to me and their grade was determined by sub-goal use, students used the sub-goals. From empirical observation, I believe it strengthened weaker students' understanding and performance.*

> *I personally think the subgoals were helpful compared to not having them. It helped students focus on the point they were supposed to take from each lesson.*

> *"Generally, I thought the subgoal work seemed to help my students to prepare better for the AP Exam. I think it guided them more slowly through the material, which was helpful for some of my less experienced students.*

In a similar theme, some of the teachers who were teaching CSP for the first time said that they appreciated having the subgoals.

> *It was helpful to have the subgoal activities in my first year teaching this course.*

> *I found the subgoals helpful as this is my first year teaching computer science principles and teaching programming.*

These comments suggest that the subgoal-oriented materials were fulfilling the purpose of the subgoal learning framework, to help novices learn problem-solving procedures by emphasizing the structure of the procedures, which novices tend to overlook. More advanced learners and more experienced teachers, however, found the subgoal comment blocks more tedious than helpful, which is the third theme from the analysis.

> *My class seems to be broken in to four groups:*
> *1. Students who used the subgoals, moved slowly, but really got it by the end*
> *2. Students who used the subgoals, relied heavily upon them, and really didn't know what they were doing but "completed the task"*
> *3. Students who didn't use the subgoals initially, did poorly, and when prompted to go back and use them were not super psyched about it but their work got better*
> *4. Students who would fly through with or without the subgoals due to prior knowledge.*

> *Students with more programming experience found them to not be helpful and slowed them down, and so it was hard [for] the teacher to show them the value in it with so much prior experience.*

> *Overall I liked the added subgoals. However, for the stronger students that really understand computer science I think it was too much extra information for them and*

almost made it more confusing. I also think that it became more reading for the students and that turned some of them off.

The pat "Orient Turtle" and "Move Turtle" were so vague as to be annoying. I urged them to edit these phrases to more meaningful phrases right from the start. Please be advised that I am a retired robotics engineer with more than a million lines of code to my name.

Recognizing the diminishing usefulness of subgoals in later problem solving, several teachers recommended fading subgoal comment blocks throughout the unit. This recommendation aligns with the subgoal learning framework and design of the unit, which prompted students to use subgoal comment blocks less in later lessons.

[Students] used the subgoals originally and most of them eventually stopped using them because they felt like it slowed them down. Now that we're in Unit 5, at least one of the students is using the comments feature but calling it "subgoals." To me that says it's useful to continue introducing subgoals early so that they see how it's helpful for organizing code.

I think the subgoals were helpful in getting students to organize their thinking, however, some students thought they were not helpful and created extra steps they saw as unnecessary. I'd like to see subgoals introduced and used early on and then as an optional piece later in the unit.

As they moved forward and understood the students stopped using the sub goals because they felt like it slowed them down.

Despite diminishing usefulness for problem solving, many teachers appreciated that the subgoal comment blocks prompted students to practice writing comments in their code.

I loved the rewritten Unit 3. Great improvement on abstraction / comments / program documentation.

Even though most students thought the subgoals were kind of "in the way", I thought they were useful to keep students focused on making good comments of their own. That is, they weren't as helpful for learning programming concepts as much as they were helpful for practicing good documentation.

I LOVED that we were using comments! I sort of wish we would have called them comments ("Use comments to define sub-goals of your program"). Some students were confused if they were required for the program to run or if they were optional. I think I needed to do more modeling of subgoals in class - especially in the unit performance task. That would have been a good time to really reinforce that skill. I will be doing that next year.

In addition to practicing using comments, some teachers appreciated that the subgoal blocks helped their students learn to organize their programs, especially as they worked on larger projects.

I think the subgoal was a nice structure to help organize their thoughts. But, when students saw that the coding for it was optional work that didn't actually affect their final code, they all opted to skip using them after the first couple instances. And honestly, while comments are really useful for organizing thoughts and communicating what is going on to other humans who read your code, students want

to simply focus on the crux of the lesson. Additional work (with no reward) seems a waste of mental energy to them.

I strongly feel my students benefited from using subgoals, although some of them found having to use them cumbersome at times. Additionally, I think the use of subgoals guides students into more organized programmers by including appropriate comments along with their code.

My co-worker and I work together in CSP. However, her students did not use the subgoals and my students did. I found that emphasizing the subgoals really rolled over to emphasizing all the comments that should be used when programming. Many of my students were in the habit of the subgoals (comments) which really helped as we began the longer and more abstract coding.

Of the 71 responses, 32 included only positive feedback about the subgoal redesign, mostly on the themes of helping less experienced or lower-performing students, helping less experienced teachers, practicing commenting, or practicing organization in large projects. Fourteen of the responses included only negative feedback, including a few about technical difficulties but mostly on the theme of being tiresome to students with prior knowledge. The remaining 25 responses either had neutral information, including four people asking for "Unit 5 – Building Apps" to include subgoal blocks, or both positive and negative information, mostly on the theme of fading the subgoal-oriented materials throughout the unit.

In addition to providing qualitative feedback, we asked teachers to complete four Likert scale (i.e., "1 – Strongly Disagree" to "5 – Strongly Agree") questions about the redesigned unit. Specifically, we wanted to know if the students found the subgoal-oriented materials or subgoal comment blocks confusing and if the students found them helpful. In addition, we asked the teachers to compare this year's students who learned with subgoals to last year's students who learned with the previous version of the unit. We asked them to compare their performance and the level of help they needed during the unit. For each question, we included an "Unsure" option to avoid a forced choice, which was excluded when calculating the mean. The distribution of responses is shown in Table 7.

Table 7. Likert Scale Questions Asked to Teachers who Used Subgoal Blocks.

Question	Str. Disagree	Disagree	Neutral	Agree	Str. Agree
My students found the subgoals in the activities to be confusing.	26%	28%	23%	22%	3%
My students found the subgoals in the activities to be helpful.	2%	14%	21%	39%	24%
This year's students needed more help during the subgoal-oriented programming unit than previous year's students have needed in the previous programming unit.	29%	34%	27%	9%	1%
This year's students performed better in the subgoal-oriented programming unit than previous year's students have performed in the previous programming unit.	0%	16%	43%	30%	11%

The first question was, "My students found the subgoals in the activities to be confusing." The mean was below the neutral point at 2.54, and only 25% teachers agreed or strongly agreed with the statement. The teachers who agreed that their students found the subgoals confusing largely overlapped with the teachers who said that their students with prior experience found the subgoal cumbersome.

The second question was, “My students found the subgoals in the activities to be helpful.” The mean was above the neutral point at 3.69, and only 16% of teachers disagreed or strongly disagreed with the statement. Many of the teachers who gave feedback that their students with little prior experience or who struggled with the concepts benefitted from the subgoals also strongly agreed that their students found the subgoals helpful. To compare this year’s subgoal redesign to the previous design, we asked teachers to rate the statement, “This year’s students needed more help during the subgoal-oriented programming unit than previous year’s students have needed in the previous programming unit.” Many teachers were teaching CSP for the first time, and 53 marked the “Unsure” option, which largely aligned to teachers who mentioned being a first-time CSP teacher in the feedback. Of the remaining teachers, the mean was below the neutral point at 2.61, and only 10% of teachers agreed or strongly agreed compared to 63% who disagreed or strongly disagreed.

To continue the comparison, we asked teachers to rate the statement, “This year’s students performed better in the subgoal-oriented programming unit than previous year’s students have performed in the previous programming unit.” Again, many teachers (58) marked the “Unsure” option, and of the remaining teachers, the mean was above the neutral point at 3.48. 41% of teachers agreed or strongly agreed that this year’s students performed better during this unit than last year’s students, and only 16% disagreed. None strongly disagreed.

Based on the qualitative and quantitative results from the teacher survey, the subgoal redesign seemed to be successful, particularly for supporting students who have less prior knowledge or struggled with the programming unit. For students who had prior knowledge of programming, however, the subgoals were tedious and likely provided no value. This finding aligns with the subgoal learning framework, which posits that subgoal-oriented instruction is most useful for absolute novices and less useful as learners gain knowledge. Therefore, students who already have experience with programming before starting CSP’s unit on programming should not be forced to use the subgoal comment blocks.

For the same theoretical reasons and based on teachers’ recommendations, the focus on subgoal learning should fade throughout the unit. However, we do not believe that eliminating the subgoal comments all together in later lessons is the best choice. In the quantitative data from AppLab about subgoal block usage, we found that students used the newer subgoal blocks “Set pen properties” and “Write a loop” when they were not required to, and many students continued to use all subgoal blocks throughout the lessons, especially during more open-ended tasks. Therefore, while we agree that extra support, such as subgoal-oriented instruction, should be faded throughout the lesson, the fading should likely be based on when each individual subgoal is introduced. Subgoal-oriented instruction can continue to be used throughout a lesson, or *even for new units or courses as the quote below suggests*, but it should be focused on only new concepts. Then all subgoal blocks can remain in the environment as a feature to be used when the student deems them useful.

> *I liked the sub-goals and am implementing that concept into my intro class and revisiting it with my AP CSA (JAVA) class.*

6 Conclusions

In this project we applied the subgoal learning framework to redesign the introduction to programming unit of Code.org’s CSP course. The redesign included adding subgoal comment blocks to the AppLab programming environment and using subgoal-oriented instruction to help students learn the subgoals of programming procedures. The subgoal-oriented instruction was faded through six lessons within the unit from prepopulating the workspace with subgoal blocks (two units) to stating the subgoals required to achieve the solution (two units) to not explicitly stating subgoals but having the blocks still available (two units).

The redesign was a novel application of the subgoal learning framework in three ways. First, subgoal-oriented instructions have not been studied in a K-12 setting before, only in higher education. Second, subgoal-oriented instruction had been used for block-based programming, but it has not been implemented as comment blocks that students can use in their code. Third, subgoal-oriented instruction had mostly been used for short instructional sections about 30-45 minutes long, and this implementation took students multiple weeks to complete.

Regarding the first novel aspect and our research question about subgoals' efficacy, we found that subgoal-oriented instruction can be effective in a K-12 setting. Students who used the subgoal redesigned unit performed better on multiple-choice questions about procedures and wrote more on the open-ended response questions than students who used the original version of the unit. In addition, many students who had access to subgoal comment blocks continued to use those comments blocks after the instructions no longer prompted them to do so. Based on the teacher survey, the subgoals were likely most useful for students who had little prior knowledge or struggled with the concepts and for teachers who had no prior experience teaching CSP. By collecting data from a large group of learners with diverse learning backgrounds, we also found that students with substantial prior knowledge found the subgoals more tiresome than useful. Prior research controlled for prior knowledge [5, 6], so this finding identifies an important condition for the effective implementation of subgoal learning—that learners need to be novices.

Regarding the second novel aspect and our research questions about subgoals' use by students and teachers' perceptions, we found that implementing the subgoals as comment blocks had both benefits and detriments. Though many students found the blocks useful enough to continue using them without being prompted to do so, many students also opposed using the blocks and would even remove them if they were prepopulated in the workspace. From the survey we heard teachers say that their advanced students found the blocks burdensome from the beginning and that most students eventually outgrew consistent use of the blocks. An important exception, however, is that block usage increased when students worked on more open-ended or collaborative programming tasks. Furthermore, many teachers said that using the subgoal blocks, even when students did not like to, helped students to practice commenting their code and helped students who were struggling to complete tasks.

Regarding the third novel aspect and our research question about subgoals' efficacy, we found that the differences between the subgoal and control groups did not change throughout the unit. In the AppLab and teacher survey data, we found that use of subgoal comment blocks decreased throughout the semester, following the fading of subgoal-oriented instructions in the lessons. This decrease in subgoal use did not have a concomitant decrease in the differences between groups on the multiple choice and open-ended response data throughout all six lessons. The multiple choice and open-ended response data likely represent formative assessments, and it is possible that after students study for summative assessments, like exams, the differences between groups would disappear. If this were the case, like it was in Margulieux et al. [8], the subgoal learning framework still has value in supporting students who might otherwise be at risk of failing or dropping out of the course by integrating additional guidance with existing instruction and activities. The goal of the subgoal learning framework is to help novices in the earliest stage of learning so that they can achieve later stages of learning, and it was successful in this case.

ACKNOWLEDGMENTS

This work is funded in part by the National Science Foundation under grant 1712231. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Robert K. Atkinson. 2002. Optimizing learning from examples using animated pedagogical agents. *Journal of Educational Psychology* 94, 2 (2002), 416.

[2] Robert K. Atkinson and Sharon J. Derry. 2000. Computer-based examples designed to encourage optimal example processing: A study examining the impact of sequentially presented, subgoal-oriented worked examples. In *Fourth International Conference of the Learning Sciences*.

[3] John Bransford. 2000. *How people learn: Brain, mind, experience, and school*. National Academies Press.

[4] Richard Catrambone. 1994. Improving examples to improve transfer to novel problems. *Memory & Cognition* 22, 5 (1994), 606–615.

[5] Richard Catrambone. 1996. Generalizing solution procedures learned from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 22, 4 (1996), 1020.

[6] Richard Catrambone. 1998. The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General* 127, 4 (1998), 355.

[7] Richard Catrambone. 2011. Task analysis by problem solving (TAPS): Uncovering expert knowledge to develop high-quality instructional materials and training. In *2011 Learning and Technology Symposium*, 132–139.

[8] CollegeBoard. 2017. AP Computer Science Principles: Course and Exam Description. Retrieved March 24, 2019 from http://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf

[9] Adrienne Decker, Lauren E. Margulieux, and Briana B. Morrison. 2019. Using the SOLO Taxonomy to Understand Subgoal Labels Effect in CS1. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 209–217. DOI:https://doi.org/10.1145/3291279.3339405

[10] W. Dick, L. Carey, and J. O. Carey. 2011. *The Systematic Design of Instruction* (8th ed.). Allyn and Bacon, New York, NY.

[11] Joentausta, Johanna, and Arto Hellas. 2018. Subgoal labeled worked examples in K-3 education. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pp. 616-621. ACM, 2018.

[12] Juho Kim, Robert C. Miller, and Krzysztof Z. Gajos. 2013. Learner sourcing subgoal labeling to support learning from how-to videos. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, 685–690.

[13] Lauren E. Margulieux and Richard Catrambone. 2014. Improving problem solving performance in computer-based learning environments through subgoal labels. In *Proceedings of the first ACM Conference on Learning@ Scale*, 149–150.

[14] Lauren E. Margulieux, Richard Catrambone, and Mark Guzdial. 2016. Employing subgoals in computer programming education. *Computer Science Education* 26, (2016), 1–24.

[15] Lauren E. Margulieux, Mark Guzdial, and Richard Catrambone. 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, 71–78.

[16] Lauren E. Margulieux, Briana B. Morrison, and Adrienne Decker. 2019. Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1. In I*nnovation and Technology in Computer Science Education Proceedings*, 7. DOI:https://doi.org/10.1145/3304221.3319756

[17] Briana B. Morrison, Adrienne Decker, and Lauren E. Margulieux. 2016. Learning Loops: A Replication Study Illuminates Impact of HS Courses. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, 221–230. DOI:https://doi.org/10.1145/2960310.2960330

[18] Briana B. Morrison, Lauren E. Margulieux, Barbara Ericson, and Mark Guzdial. 2016. Subgoals Help Students Solve Parsons Problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 42–47. DOI:https://doi.org/10.1145/2839509.2844617

[19] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. 2015. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, 21–29. DOI:https://doi.org/10.1145/2787622.2787733

[20] Y. Qian, S. Hambrusch, A. Yadav, and S. Gretter. 2018. Who needs what: Recommendations for designing effective online professional development for computer science teachers. J*ournal of Research on Technology in Education* 50, 2 (2018), 164–181.