2015

Subgoals Help Students Solve Parsons Problems

Briana B. Morrison
University of Nebraska at Omaha, bbmorrison@unomaha.edu

Lauren E. Margulieux
Georgia Institute of Technology

Barbara Ericson
Georgia Institute of Technology

Mark Guzdial
Georgia Institute of Technology

Subgoals Help Students Solve Parsons Problems

Briana B. Morrison
School of Interactive Computing
Georgia Institute of Technology
85 5th Street NW
Atlanta, GA, 30332-0760
bmorrison@gatech.edu

Lauren E. Margulieux
School of Psychology
Georgia Institute of Technology
654 Cherry Street
Atlanta, GA, 30332-0170
l.marg@gatech.edu

Barbara Ericson, Mark Guzdial
School of Interactive Computing
Georgia Institute of Technology
85 5th Street NW
Atlanta, GA, 30332-0760
ericson, guzdial@cc.gatech.edu

ABSTRACT
We report on a study that used subgoal labels to teach students how to write `while` loops with a Parsons problem learning assessment. Subgoal labels were used to aid learning of programming while not overloading students' cognitive abilities. We wanted to compare giving learners subgoal labels versus asking learners to generate subgoal labels. As an assessment for learning we asked students to solve a Parsons problem – to place code segments in the correct order. We found that students who were given subgoal labels performed statistically better than the groups that did not receive subgoal labels or were asked to generate subgoal labels. We conclude that a low cognitive load assessment, Parsons problems, can be more sensitive to student learning gains than traditional code generation problems.

Categories and Subject Descriptors
Social and professional topics~Computer science education

General Terms
Measurement, Design, Experimentation.

Keywords
Subgoal labels, Cognitive Load, Contextual Transfer, Parsons problem.

1. INTRODUCTION
As educators we want to simplify the learning process to present only what is germane to make student learning efficient. As researchers we want to find empirical evidence for effectiveness. One proven method for enhancing learning is to reduce unnecessary cognitive load on the student while they are trying to learn to solve problems [26]. There are several ways to reduce cognitive load, including using worked examples [16].

Worked examples typically include a problem statement along with a step-by-step procedure for solving the problem. Worked examples are most effective when used in worked example-practice pairs [2]. In these pairs, students study a worked example solution and immediately practice by solving a similar problem.

Segmenting worked examples and including subgoal labels have also been shown to be effective in improving learning [2]. Segmenting includes separating portions of the worked example to isolate each step in the process [27]. Subgoal labels are names given to a set of steps in the solution process allowing the user to "chunk" the information to ease learning [10].

SIGCSE '16, March 2–5, 2016, Memphis, TN, USA.

ACM 978-1-4503-3685-7/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2839509.2844617

While these cognitive load reducing techniques have been empirically tested in math and science disciplines, we have been the first to test these with computer science learning [17]. Margulieux et al. [17] demonstrated learning benefits for subgoal labels with a drag-and-drop programming language. We continued this work to show the effectiveness with textual programming languages [21]. This paper reports on use of a new assessment to measure students' learning.

In our previous experiment [21], we created instructional material to teach introductory programming students about the process of using and writing a `while` loop to solve programming problems. There were three treatment conditions: (1) *no* subgoal labels provided, (2) subgoal labels *given* for each segment of instructions, and (3) subgoal labels *generated*, in which students were asked to generate their own labels for groups of solution statements. Within each treatment group, participants were randomly assigned to either an *isomorphic* or *contextual transfer* group. In the isomorphic transfer group, the problem to be solved in the worked example-practice problem pair was identical to the worked example in both procedural steps and cover story. The differences were the values of the numbers to be calculated. In the contextual transfer group, the problem to be solved in the worked example-practice problem pair involved the same procedural steps but the cover story and numeric values changed. Participants' learning was measured several different ways. In [21] we reported on assessment using novel problem solving tasks by writing code and a post-test. In this paper, we present the results from the Parsons problem assessment and discussion the implications.

Our research questions associated with the experiment were based on previous research on subgoal labels:

- Would students who generated subgoal labels learn better than those who were given the subgoal labels, and would both groups would do better than those who had no subgoals at all?
- What effect would contextual transfer have on student performance? Would those who generated subgoal labels take the least amount of time on the assessment?

2. BACKGROUND
In this section we review the current literature for cognitive load, worked examples, subgoal labeling, and our assessment technique, Parsons problems.

2.1 Cognitive Load
Cognitive load can be defined as the load imposed on an individual's working memory by a particular learning task [32]. The cognitive load required to comprehend materials directly affects how much students learn and affects their performance

scores on assessments related to that task [16]. If students have to keep too many things in working memory in order to understand a concept, learning suffers. As designers of instructional material, it is our responsibility to ensure that we do not overload the learner's working memory where possible when presenting new material. We should ensure that students' attention is directed at what is germane learning material rather than extraneous aspects.

The central problem identified by Cognitive Load Theory (*CLT*) is that learning is impaired when the limited capacity of working memory is exceeded [24]. Currently CLT [19, 28, 30] defines two different types of cognitive load on a student's working memory: intrinsic load and extraneous load.

Intrinsic load is a combination of the innate difficulty of the material being learned as well as the learner's characteristics [15]. Extraneous load is the load placed on working memory that does not contribute directly toward the learning of the material---for example, the resources consumed while understanding poorly written text or diagrams without sufficient clarity [15]. Working memory resources that are devoted to information that is relevant or germane to learning are referred to as 'germane resources' [29].

The intrinsic and extraneous loads can be controlled through instructional design. When designing instructional material care should be given to eliminate any possible extraneous load while attempting to minimize the intrinsic load. It is believed that worked examples, when carefully designed, can accomplish both of these goals [24].

2.2 Worked Examples

Worked examples are one type of instruction used to teach students a procedural process for problem solving. Worked examples give learners concrete examples of the procedure being used to solve a problem.

Eiriksdottir and Catrambone argue that learning primarily from worked examples does not inherently promote deep processing of concepts [13]. While it may result in better initial performance because examples are more easily mapped to similar problems, it is less likely result in retention and transfer [13]. When studying examples, learners tend to focus on incidental features rather than the fundamental features because incidental features are easier to grasp and novices do not have the necessary domain knowledge to recognize fundamental features of examples [11]. For example, when studying physics worked examples, learners are more likely to remember that the example has a ramp than that the example uses Newton's second law [11]. A focus on incidental features leads to ineffective organization and storage of information that, in turn, leads to ineffective recall and transfer [6].

2.3 Subgoal Labels

To promote deeper processing of worked examples and, thus, improve retention and transfer, worked examples have been manipulated to promote subgoal learning. Subgoal learning refers to a strategy used predominantly in STEM fields that helps students deconstruct problem solving procedures into subgoals, functional parts of the overall procedure, to better recognize the fundamental components of the problem solving process [1]. Subgoals are the building blocks of procedural problem solving and they are inherent in all procedures except the most basic.

Subgoal labeling is a technique used to promote subgoal learning that has been used to help learners recognize the fundamental structure of the procedure being exemplified in worked examples [8–10]. Subgoal labels are function-based instructional explanations that describe the purpose of a subgoal to the learner. For example, in the problem in Figure 1 for the first two lines of code the subgoal label might read "Initialize Variables." This label provides information about the purpose of that subgoal and the function behind the steps within it. Studies [3, 4, 8–10, 17, 18] have consistently found that subgoal-oriented instructions improved problem solving performance across a variety of STEM domains, such as block-based programming (e.g., [17]) and statistics (e.g., [10]).

Studies have found that giving subgoal labels in worked examples improves student performance when they are solving novel problems without increasing the amount of time they spend studying instructions or working on problems (e.g., [17]). Subgoal labels are believed to be effective because they visually group the steps of worked examples into subgoals and meaningfully label those groups [1]. This format highlights the structure of examples, helping students focus on structural features and more effectively organize information [2].

By helping learners organize information and focus on structural features of worked examples, subgoal labels are believed to reduce the extraneous cognitive load that can hinder learning but is inherent in worked examples [25]. Worked examples introduce extraneous cognitive load because they are necessarily specific to a context, and students must process the incidental information about the context even though it is not relevant to the underlying procedure [30]. Subgoal labels can reduce focus on these incidental features by highlighting the fundamental features of the procedure [25]. Subgoal labels further improve learning by reducing the intrinsic load by providing a mental organization for storing information.

Subgoal labels that are independent from a specific context have been the most effective type of subgoal labels in the past [7, 10]. Catrambone found that learners who were given labels that were abstract (e.g., Ω) and had sufficient prior knowledge performed better than those who were given labels that were context-specific (e.g., isolate x) on problem solving tasks done after a week long delay or in problems that required using the procedure differently than demonstrated in the examples [10]. Catrambone explained this exception by arguing that learners with sufficient prior knowledge were able to correctly explain to themselves the purpose of the subgoal and that by self-explaining the function of the subgoal--the self-explaining presumably due to the abstract label--was more effective than providing labels.

In summary, previous research has found that learners who generate subgoal labels (or self-explain) learn more than those who are given subgoal labels [8, 10]. Additionally those who are given subgoal labels learn more than those who have no subgoal labels. Learners who are given more abstract subgoal labels for problems perform better than those who are given context specific labels. Learners who generate subgoal labels take more time during the learning phase and less time during the assessment phase than those who are given subgoal labels or those with no subgoal labels. Given subgoal labels are considered a lower cognitive load than generating subgoal labels, though both are a higher cognitive load than no subgoal labels; however, no subgoal labels represent no additional learning instructions or cognitive aids for the student. These findings provide the basis for our hypotheses.

2.4 Parsons Problems

One way to make the learning of programming more efficient and effective is to reduce the amount of time that learners struggle with syntax errors. One approach is to use Parsons problems [23] in which correct code is broken into code fragments that have to

be put in the correct order with the correct indention. There are several variants of Parsons problems such as including unnecessary code as distractors [12].

Work in this area [12] has found that Parsons problems scores significantly correlate with code writing scores. Parsons problems are simpler than writing code, e.g., students cannot get syntax errors. It has a lower cognitive load because students do not have to focus on issues like syntax while practicing meaning and sequencing within problem solving. This means that Parsons problems might be a more efficient way to practice than the traditional approach, hours of writing code.

2.5 Hypotheses

In this study we sought to combine subgoal labels with a Parsons problem assessment to determine if the performance gains found with subgoal labels still apply with a different type of assessment and if the relative performance speed replicated previous studies.

The contextual transfer was intended to promote deep learning instead of superficial learning as the contextual transfer groups had to do non-superficial transfer during learning [5, 13, 22]. However transfer would not necessarily improve performance on Parsons problems because learners do not have to determine how to apply a conceptual understanding of the procedure to a specific problem since the lines of code in the Parsons problem are provided for them. We entered into the study with the following hypotheses:

H1. Participants who learn with subgoal labels (given or generated) will perform better on low cognitive load assessments.

H2. Changing the context or "cover story" between the worked example and practice problem should have limited effect on student performance on Parsons problem assessments.

H3. Those who generate their own subgoal labels will take less time on assessments than those who are given subgoals or receive no subgoals.

3. METHOD OF STUDY

3.1 Purpose

Participants in introductory programming classes were given instructional material designed to teach them to solve programming problems using `while` loops. This common introductory programming task requires only minimal prior programming knowledge (arithmetic operations and Boolean expressions) to complete at a basic level. This paper reports on data gathered during [21], with additional data collected during the summer of 2015. In this paper we provide only the differences from the study method presented in the original paper.

Participants were recruited from 7 different introductory programming courses at two technical universities in the Southeast United States. At one institution the study was conducted over a two week period; at the other institution the study was done over a month period. Because the courses teach different programming languages (see Table 1), pseudo-code was used in the task to make it independent from any one programming language.

Pseudo-code is relatively easy for programmers to understand regardless of the programming languages that they know [31]. The study was conducted in either a closed lab setting with up to 30 computers in a single room (one institution) or completely through email and over the internet (second institution). Students were given an explanation of the study. They worked independently. The sessions typically lasted between 1 and 2 hours, depending on the rate at which participants completed the tasks.

Table 1. Classes Participating in Study

Programming Language	Majors
C++ or MATLAB	Engineering
C#	Game Development
Java or Python	Computer Science, Information Technology, Software Engineering, Non-Majors

3.2 Instructional Materials

The instructional materials were the same as those used in [21]. Participants were given three interleaved worked examples and practice problems. The worked examples came in three formats, which varied between participants. The first format contained no subgoal labels. The second format grouped steps of the example by subgoal and provided meaningful subgoal labels for each group as is typical in subgoal label research (e.g.,[17]). The third format grouped steps of the example by subgoal and provided a spot for participants to write generated subgoal labels for each group. Exampes of all three formats can be seen in Figure 1. Participant groups were also divided into a *contextual transfer* group or an *isomorphic* problem group. In the *isomorphic* group the "cover story" stayed the same for the worked example and practice problem, only the data values changed. In the *contextual transfer* group the "cover story" between the worked example and practice problem changed. In other words, if the worked example was for averaging tip amounts, the practice problem would involve averaging grades.

No labels	Given Labels	Generate Labels

sum = 0
lcv = 1
 	
WHILE lcv <= 100 DO
 	
sum = sum + lcv
 	
lcv = lcv + 1
ENDWHILE
	Initialize Variables
	sum = 0
	lcv = 1
	Determine Loop Condition
	WHILE lcv <= 100 DO
	
	Update Loop Var
	lcv = lcv + 1
	ENDWHILE
		Label 1: _____
		sum = 0
		lcv = 1
		Label 2: _____
		
		WHILE lcv <= 100 DO
		
		Label 3: _____
		lcv = lcv + 1
		ENDWHILE

Figure 1. Partial worked example formatted with no labels, given labels, or placeholders for generated labels.

After completing the instructions, participants completed a Parsons problem to measure their problem solving performance.

3.3 Design

The experiment was a 3-by-2, between-subjects, factorial design: the format of worked examples (unlabeled, subgoal labels given, or subgoal labels generated) was crossed with the transfer distance between worked examples and practice problems (isomorphic or contextual transfer). The dependent variables were performance on the pre- and post-test, problem solving task, and time on task.

3.4 Participants

Participants were 119 students from two technical universities in the Southeast United States (Table 2). Students were offered credit for completing a lab activity or extra credit as compensation for participation. All students from these courses were allowed to participate, regardless of prior experience with programming or using `while` loops. To account for prior experience, participants were asked about their prior programming experience in high school (either regular or advanced placement courses) and college and whether they had experience using `while` loops. Other

demographic information collected included gender, age, academic major, high school grade point average (GPA), college GPA, number of years in college, reported comfort with use of a computer, expected difficulty of the programming task, and primary spoken language. There were no statistical differences between the groups for demographic data, which is expected because participants were randomly assigned to treatment groups. Participants also took a multiple-choice pre-test to measure problem solving performance for using `while` loops. Average scores on the pre-test were low, 1.6 out of 5 points, with 23% (28 out of 119) of participants earning no points.

Table 2. Participant Demographics

Age	Gender	GPA	Major
M = 21.6	71% male	*M* = 3.2/4	52% CS major

Many participants did not complete all tasks of the experiment. Participants received compensation regardless of the amount of time or effort that they devoted to the experiment, which might have caused low motivation in some participants. Participants who did not attempt all tasks were excluded from analysis. Participants who answered more than two questions correctly out of the five on the pre-test were excluded from analysis because the instructions were designed for novices.

3.5 Procedure

An outline of the entire study is given in [21]. Briefly, particiants completed a demographic questionnaire and pre-test. This was followed by the instructional period and a 10 item survey designed to measure cognitive load [20].

Once participants completed the cognitive load survey, they started the assessment period which included three types of tasks. Only the Parsons problem assessment task will be discussed here. (See [21] for a complete description and analysis of the initial assessment task, the cognitive load measurement and post test.) The Parsons problem used for assessment was a version of the "rainfall problem" [14]. The problem had 13 different code pieces with between 1 and 3 lines of code in each code piece. The participants were asked to put the code pieces in order with no consideration of indentation. In other words, they indicated the order of the code segments by numbering them. After the assessment period, participants completed a post-test.

Throughout the procedure we recorded the time taken to complete each task. We collected process data throughout the instructional period and performance on the training activities and practice problems to ensure that participants were completing tasks.

4. ANALYSIS AND RESULTS

We scored participants' Parsons problem answers for correct order to create their score. Participants ranked the 13 code pieces from the Parsons problem and we gave them one point for each code piece that was in the correct order relative to the pieces around it. For example, if participants ranked the 4th, 5th, and 6th pieces of the problem as the 5th, 6th, and 7th pieces of their solution, they would receive two out of three possible points for those three pieces. The first piece would be counted as wrong because it is not following the 3rd piece, but the other two pieces would be counted as correct, as they are following the correct piece. This scoring scheme captures participants' understanding than scoring for absolute correct order as it does not penalize correct sequences of code that follow incorrect sequences.

4.1 Accuracy

The effect of the interventions on Parsons problem performance depended on the worked example manipulation. Participants who were given subgoal labels in the worked example performed better than those who generated their own labels or were not given labels, F (2, 113) = 3.8, MSE = 10.6, p = .026, est. ω^2 = .07, f = .18 (see Figure 2). We found no main effect of transfer distance, F (2, 113) = 1.1, MSE = 10.6, p = .303, est. ω^2 = .009. We also found no interaction between worked example format and transfer distance, F (2, 113) = 0.07, MSE = 10.6, p = .937, est. ω^2 = .001.

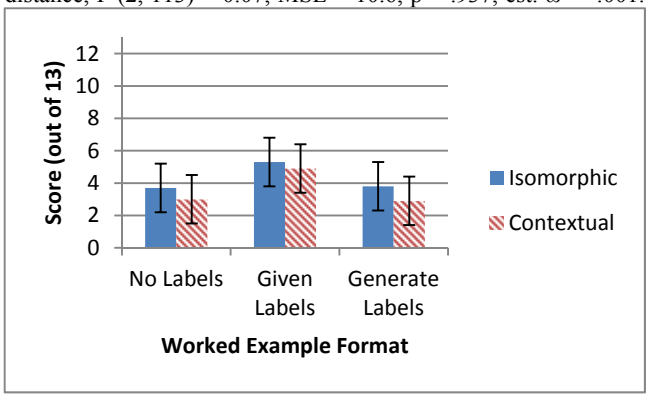

Figure 2. Parsons Problem performance graphed with worked example format on the x-axis, transfer distance as separate colors, and score on the y-axis.

4.2 Time Efficiency

Time spent on the Parsons problem task differed among levels of the worked example manipulation. Participants who generated their own subgoal labels in the worked example (*M* = 2.7 minutes) completed the task faster than those who were not given labels (*M* = 4.2 minutes), F (2, 113) = 4.8, MSE = 5.2, p = .010, est. ω^2 = .07, f = .20 (see Figure 3). We found no main effect of transfer distance, F (2, 113) = 2.2, MSE = 5.2, p = .142, est. ω^2 = .02. We found no interaction between worked example format and transfer distance, F (2, 113) = 2.1, MSE = 5.2, p = .126, est. ω^2 = .03.

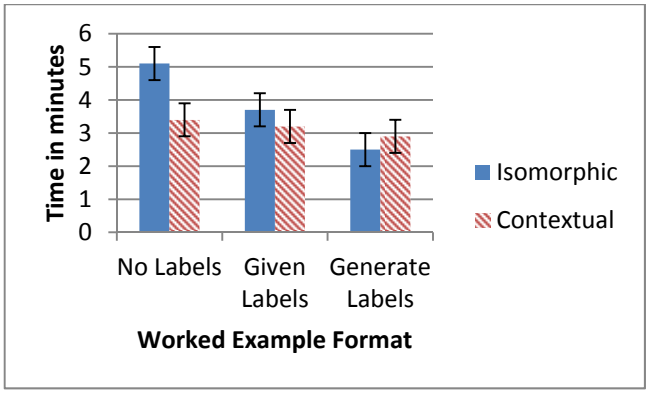

Figure 3. Parsons Problem time graphed with worked example format on the x-axis, transfer distance as separate colors, and minutes on the y-axis.

5. DISCUSSION

In this section we summarize our findings related to our original hypotheses and discuss the implications for teaching.

5.1 Findings

In this study, the results reinforce some of the previous subgoal research. We 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 other words, both the Given-Isomorphic and

Given-Contextual Transfer groups performed statistically better than the other groups. In all cases the isomorphic groups did better than their contextual transfer counterpart, however these differences were not significant. We thus have partial support for our first hypothesis: Participants who learn with subgoal labels (given *only*) will perform better on low cognitive load assessments. Because there were no statistical differences between the isomorphic problem groups and the contextual transfer groups, we have support for our second hypothesis, transfer appears to have limited effect on student performance on this task.

The groups that generated their own subgoal labels completed the Parsons problem assessment in statistically less time than those who received no subgoal labels. This is consistent with previous research that indicates those who generate their own subgoal labels can recall the learned information more quickly. However this group did not perform the best, as would have been predicted by previous research. We are not sure why this is the case. One hypothesis is that this group "knew what they knew" and what they did not. They did not want to waste time puzzling out a solution if they predicted they would be unsuccessful. It may also be due to survey fatigue. The Generate groups took the most time during the instructional period (as expected) and the Parson problem assessment was near the end of the study. They may have reached their tolerance level and just wanted to finish.

We would expect the None-Isomorphic group to take the most time on the assessment as they most likely had the most shallow learning. This hypothesis agrees with our findings. There were no statistically significant differences based on the programming language used in the class.

5.2 Implications

Participants that were given subgoal labels performed overall better than those that did not have subgoal labels and those that generated their own subgoal labels. Though participants in the generate labels and no labels conditions performed equally, participants who generated their own labels completed the task faster than those who did not receive labels.

What can we learn from this study? It appears that learning which occurs with a low cognitive load can be assessed with low cognitive load assessments. In our previous work we found that low cognitive load learning did not always lead to better learning performance on high cognitive load assessments like writing code [21]. We can take two implications from these findings.

First, subgoal labels can reduce cognitive load which allows student to focus and learn more efficiently. Students who are given subgoal labels while learning problem solving can most likely recall those labels when needed to arrange code segments into order – the order of the learned subgoal labels. In previous work we have shown that learning subgoal labels also helps with transfer [17]. The subgoal labels provide structure for organization of student learning.

Giving subgoal labels to students may be more beneficial than having them generate their own subgoal labels for two reasons. First, giving the students the subgoal labels requires less time on their part for the learning acquisition phase. It is much quicker to read labels and learn their pattern than generating their own labels. Second, unless the instructor is reviewing and correcting the generated labels, student misconceptions may persist. Students may generate labels that are too context specific and do not transfer to other problems. We saw some evidence of this within the generated labels of the students. This may also explain why the Given Labels groups outperformed the Generate Label groups.

As instructors we should consider providing students with subgoal labels which are consistent across problems – initialization of variables, determining the loop termination / continuation condition, updating the loop control variable, etc. Students who learn the "pattern" of the problem solution can then recall the pattern when asked to order code segments.

Second, we should use more Parsons problems as low cognitive load assessments, either formative or summative, for students. Students who have learned may fail at high cognitive load assessments simply because the cognitive load is too great for them to succeed. Low cognitive load assessments have greater sensitivity. Parsons problems allow students to demonstrate that they understand the meaning and sequence of programs without having to also generate syntax. In summary, if cognitive load is increased in either the learning activity or the assessment activity student performance can suffer.

Previous research found that the transfer condition had a statistically significant effect on the performance of a code writing assessment [21]. Yet it had limited effect on the results of this study. This may be explained by cognitive load. We know that adding transfer between the worked example and practice problem introduces additional cognitive load--students must do non-superficial transfer during learning. This additional cognitive load could alter the learning just enough so that it shows up in a high cognitive load task (writing code from scratch) but not in a low cognitive load task (Parsons problems). Students who can learn the appropriate order of the subgoals may be able to demonstrate that knowledge on Parsons problems, where they may not be able to do so when writing code from scratch [21].

6. CONCLUSION

Research in educational psychology using subgoals in other disciplines indicate that generating subgoals results in deeper learning than given subgoals which results in more learning than receiving no subgoals. In [21] we found partial support replicating these findings. This study found that computer science students who are given subgoal labels (a low cognitive load activity) can perform better on a low cognitive load assessment. They statistically outperform the group who were required to generate subgoal labels and the group that received no subgoal labels. This study provides more evidence that learning programming is inherently different than learning physics or statistics – the educational psychology principles somewhat apply, but are not completely replicable. We believe the answer lies in the cognitive load required to learn programming. Adding even one additional piece to the learning puzzle (contextual transfer, generating subgoals) can have significant effects on learning performance.

Another aspect to consider is student time in both learning and completing the assessments. We found that those who were asked to generate subgoal labels took statistically more time to complete the instructional tasks [21] but completed the assessment task significantly quicker than the other groups. The happy medium may lie with the group that was given subgoal labels. They did not take significantly longer to complete the instructional material or the assessment tasks.

The interventions for this study are strongly grounded in instructional design theory and they were also applied in an authentic educational setting with an authentic educational task. Therefore, we expect that the internal and external validity of this work is high. However, because this study is the first experiment to use this type of task and because the results were different than

previous work with subgoal labels, research to replicate these results is needed to ensure the validity of this work.

7. ACKNOWLEDGMENTS

We would like to thank the students who participated in the study and their instructors who graciously gave us the time. This work is funded in part by the National Science Foundation under grant 1138378. 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.

8. REFERENCES

[1] Atkinson, R.K. et al. 2003. Aiding Transfer in Statistics: Examining the Use of Conceptually Oriented Equations and Elaborations During Subgoal Learning. *Journal of Educational Psychology*. 95, 4 (2003), 762.

[2] Atkinson, R.K. et al. 2000. Learning from examples: Instructional principles from the worked examples research. *Review of educational research*. 70, 2 (2000), 181–214.

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

[4] Atkinson, R.K. and Derry, S.J. 2000. Computer-based examples designed to encourage optimal example processing: A study examining the impact of sequentially presented, subgoal-oriented worked examples. (2000).

[5] Bjork, R.A. 1994. Memory and metamemory considerations in the training of human beings. *Metacognition: Knowing about Knowing*. MIT Press.

[6] Bransford, J.D. et al. 2000. *How People Learn: Brain, Mind, Experience, and School*. National Academy Press.

[7] Catrambone, R. 1995. Aiding subgoal learning: Effects on transfer. *Journal of educational psychology*. 87, 1 (1995), 5.

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

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

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

[11] Chi, M.T. et al. 1989. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science*. 13, 2 (1989), 145–182.

[12] Denny, P. et al. 2008. Evaluating a new exam question: Parsons problems. *Proceeding of the Fourth international Workshop on Computing Education Research* (Sydney, Australia, 2008), 113–124.

[13] Eiriksdottir, E. and Catrambone, R. 2011. Procedural instructions, principles, and examples how to structure instructions for procedural tasks to enhance performance, learning, and transfer. *Human Factors: The Journal of the Human Factors and Ergonomics Society*. 53, 6 (2011), 749–770.

[14] Johnson, W.L. and Soloway, E. 1985. PROUST: Knowledge-based program understanding. *Software Engineering, IEEE Transactions on*. 3 (1985), 267–275.

[15] Leppink, J. et al. 2013. Development of an instrument for measuring different types of cognitive load. *Behavior research methods*. 45, 4 (2013), 1058–1072.

[16] Leppink, J. et al. 2014. Effects of pairs of problems and examples on task performance and different types of cognitive load. *Learning and Instruction*. 30, (2014), 32–42.

[17] Margulieux, L.E. et al. 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. *Proceedings of the ninth annual international conference on International computing education research* (2012), 71–78.

[18] Margulieux, L.E. and Catrambone, R. 2014. Improving problem solving performance in computer-based learning environments through subgoal labels. *Proceedings of the first ACM conference on Learning@ scale conference* (2014), 149–150.

[19] van Merriënboer, J.J. and Sweller, J. 2005. Cognitive load theory and complex learning: Recent developments and future directions. *Educational psychology review*. 17, 2 (2005), 147–177.

[20] Morrison, B.B. et al. 2014. Measuring cognitive load in introductory CS: adaptation of an instrument. *Proceedings of the tenth annual conference on International computing education research* (2014), 131–138.

[21] Morrison, Briana B. et al. 2015. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. *ICER 2015* (Aug. 2015).

[22] Palmiter, S. and Elkerton, J. 1993. Animated demonstrations for learning procedural computer-based tasks. *Human-Computer Interaction*. 8, 3 (1993), 193–216.

[23] Parsons, D. and Haden, P. 2006. Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (Darlinghurst, Australia, Australia, 2006), 157–163.

[24] Plass, J.L. et al. 2010. *Cognitive load theory*. Cambridge University Press.

[25] Renkl, A. and Atkinson, R.K. 2002. Learning from examples: Fostering self-explanations in computer-based learning environments. *Interactive learning environments*. 10, 2 (2002), 105–119.

[26] Renkl, A. and Atkinson, R.K. 2003. Structuring the transition from example study to problem solving in cognitive skill acquisition: A cognitive load perspective. *Educational psychologist*. 38, 1 (2003), 15–22.

[27] Spanjers, I.A. et al. 2012. Segmentation of worked examples: Effects on cognitive load and learning. *Applied Cognitive Psychology*. 26, 3 (2012), 352–358.

[28] Sweller, J. et al. 1998. Cognitive architecture and instructional design. *Educational psychology review*. 10, 3 (1998), 251–296.

[29] Sweller, J. et al. 2011. *Cognitive load theory*. Springer.

[30] Sweller, J. 2010. Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational psychology review*. 22, 2 (2010), 123–138.

[31] Tew, A.E. and Guzdial, M. 2011. The FCS1: a language independent assessment of CS1 knowledge. *Proceedings of the 42nd ACM technical symposium on Computer science education* (2011), 111–116.

[32] van Gog, Tamara and Paas, Fred 2012. Cognitive Load Measurement. *Encyclopedia of the Sciences of Learning*. Springer.