Learning Loops: A Replication Study Illuminates Impact of HS Courses

Briana B. Morrison
College of Information Science & Technology
University of Nebraska at Omaha
6001 Dodge Street
Omaha, NE 68182
bbmorrison@unomaha.edu

Adrienne Decker
School of Interactive Games and Media
Rochester Institute of Technology
152 Lomb Memorial Drive
Rochester, NY, 14623
adrienne.decker@rit.edu

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

ABSTRACT
A recent study about the effectiveness of subgoal labeling in an introductory computer science programming course both supported previous research and produced some puzzling results. In this study, we replicate the experiment with a different student population to determine if the results are repeatable. We also gave the experimental task to students in a follow-on course to explore if they had indeed mastered the programming concept. We found that the previous puzzling results were repeated. In addition, for the novice programmers, we found a statistically significant difference in performance based on whether the student had previous programming courses in high school. However, this performance difference disappears in a follow-on course after all students have taken an introductory computer science programming course. The results of this study have implications for how quickly students are evaluated for mastery of knowledge and how we group students in introductory programming courses.

Keywords
Subgoal labels, Cognitive Load, Contextual Transfer.

1. INTRODUCTION
In the nascent field of computer science education, we have become particularly good at two things: publishing new studies and establishing that our students don't know what we think they should know. Looking at any recent conference proceedings in the discipline, you would be hard pressed to find any papers that are replication studies. And we have many examples, from Soloway [36] to McCracken [26], of our students performing below our expectation levels on supposedly relatively easy programming tasks. This paper tackles both of these issues directly: we present the results of a replication study done with a new population of students at a different institution and provide evidence that students really do learn programming constructs, but perhaps not as quickly as we, the educators, would hope.

A recent study presented results on the effectiveness of subgoal labeling with students in introductory programming courses learning to write `while` loops to solve programming problems [28]. 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 [10–12]. Subgoal labels are function-based instructional explanations that describe the purpose of a subgoal, or functional piece of the problem solution, to the learner. Some of results presented in the original study agreed with previous subgoal label research: learning with subgoal labels produces higher learning gains and better problem solving performance than learning without subgoal labels. The study compared students who learned from unlabeled worked examples (i.e., conventional examples) to those who learned from worked examples labeled with subgoal labels created by the experimenters and worked examples that prompted students to generate their own subgoal labels.

Also tested in the original study was the effect of transfer between worked examples and practice problems. Problem sets of worked example-practice problems came in two varieties: either *isomorphic* or *contextual transfer*. In the isomorphic transfer group the problem to be solved in the worked example-practice problem (WE-PP) pair was identical to the worked example in both procedural steps and cover story (i.e., context). The only thing changed was the actual values of the numbers to be calculated. In the contextual transfer group, the problem to be solved in the WE-PP pair involved the same procedural steps but the cover story and numeric values changed.

Both the unlabeled groups and the generate subgoal labels groups performed as expected in the code writing task whether or not they were in the isomorphic or contextual transfer treatment groups. In both of these cases, the isomorphic group outperformed the contextual transfer group, which cognitive load theory [27, 37, 39] would predict. Cognitive load theory (CLT) suggests that having an additional piece to figure out – the contextual transfer – may overload the cognitive processing of novices resulting in poorer performance on the assessment task. However, perhaps the most puzzling result came from the group of students who were given subgoal labels. In this treatment, the students who received problems with contextual transfer performed statistically better than the group of students receiving the isomorphic problems. This outcome was unexpected, and we are still exploring why the group receiving subgoal labels made by the experimenters would behave differently than the other groups.

To explore whether the original study was an anomaly, we set out to replicate the study with a different population at a different institution. In the original study, the average score on the post-test, which comprised items from the AP CS test, was only 31%, indicating that the students had not learned very much from the intervention. Perhaps the assessment tasks were too difficult and once again we were asking more from our students than they were capable of (a la McCracken [26]). To test this hypothesis we also asked students in a follow-on programming course to participate

ICER '16, September 8–12, 2016, Melbourne, VIC, Australia.

ACM 978-1-4503-4449-4/16/09…$15.00.
DOI: http://dx.doi.org/10.1145/2960310.2960330

in the study to compare their performance with those in the introductory course. We found that the original results were not an anomaly – they repeated. We also found that students in the follow-on course performed better on the assessments (than the introductory students) indicating that writing `while` loops can be mastered. In addition, we found that students in the introductory programming course who had taken computing courses in high school performed better than those who did not. However, this difference in performance due to high school experience was not present for students in the follow-on course.

2. BACKGROUND

This section reviews the details of the previous study and examines current literature for the effect of previous coursework on programming performance in novices.

2.1 Previous Study

The original study [28] tested hypotheses related to whether using subgoal labels to teach a programming construct would produce results similar to those achieved in other disciplines. The study proposed that using subgoal labels to help students learn would reduce the cognitive load imposed in learning.

2.1.1 Cognitive Load Theory

Cognitive load can be defined as "the load imposed on an individual's working memory by a particular (learning) task" [40]. The cognitive load required to comprehend materials directly affects how much students learn, and affects their performance scores on assessments related to that task. If students have to keep too many things in working memory in order to understand a concept, learning suffers. The central problem identified by Cognitive Load Theory (CLT) is that learning is impaired when the total amount of processing requirements exceeds the limited capacity of working memory [31]. Currently CLT [27, 37, 39] 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 previous knowledge [21]. Extraneous load is the load placed on working memory that does not contribute directly toward the learning of the material. For example, the extra resources consumed to understand poorly written text or diagrams without sufficient clarity contributes to extraneous cognitive load [21].

The intrinsic and extraneous loads can be controlled through instructional design and care should be given to eliminate any possible extraneous load while attempting to optimize the level of intrinsic load so that the learner is challenged but not overly so. It is believed that worked examples, when carefully designed, can accomplish both of these goals [24].

2.1.2 Worked Examples

Worked examples are one type of instruction used to teach procedural process to students for problem solving activities. 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 [15]. While learning from worked examples may result in better initial performance, it is less likely result in the retention and transfer [15]. When studying examples, learners tend to focus on incidental features, like those necessary for the cover story of the problem, rather than the fundamental features, like the concepts required to solve the problem. This happens because incidental features are easy to grasp and novices do not have the necessary domain knowledge to recognize fundamental features of examples [13]. A focus on incidental features leads to ineffective organization and storage of information that, in turn, leads to ineffective recall and transfer [8].

2.1.3 Subgoal Labeling

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].

Subgoal labeling is a technique used to promote subgoal learning. Studies [3, 4, 10–12, 24, 25] have consistently found that subgoal-oriented instructions improved problem solving performance across a variety of STEM domains, such as programming [24] and statistics [12]. Studies have also found that giving subgoal labels in worked examples improves performance while solving novel problems without increasing the amount of time learners spend studying instructions or working on problems [24]. 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 [32]. 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 [39]. Subgoal labels can reduce focus on these incidental features by highlighting the fundamental features of the procedure [32]. Subgoal labels further improve learning by reducing the intrinsic load by providing a mental organization (i.e., subgoals) for storing information.

2.1.4 Results

The original study tested its hypotheses by dividing the participants into three treatment groups, each with its own instructional materials: learning with no subgoal labels, learning with given pre-defined subgoal labels, and asking participants to generate their own subgoal labels after some initial training. Each treatment group was then subdivided into two sections: *isomorphic* or *contextual transfer*. In the isomorphic transfer condition, the procedure and context used to solve the WE-PP were exactly the same but the exact values in the problem changed. For example, if a worked example asked participants to find the average of quiz scores with values 70, 80, and 90, then the practice problem asked participants to find the average of quiz scores with values 75, 85, and 95. In the contextual transfer condition, the procedure used to solve the WE-PP were the same except the context of the problem changed. For example, if a worked example asked participants to find the average of quiz scores, then the practice problem asked participants to find the average of money amounts.

The original study found that, similar to previous research in other disciplines, students who learned with subgoal labels (either given or generated) performed better on the code writing assessments than those who learned without subgoal labels. The participants who generated their own subgoal labels did not perform better on the code writing tasks requiring transfer, unlike previous research.

The unexpected results occurred with the given subgoal label

group. Cognitive Load Theory would predict that learning with given subgoal labels and no contextual transfer should impose lower cognitive processing than learning with given subgoal labels *and* contextual transfer. The contextual transfer would require additional working memory to process. However the results from the original study directly contradict this. The original study found, unlike the other two treatment groups, that the participants that learned with given subgoal labels *and* contextual transfer significantly outperformed the given subgoal labels with isomorphic problems. This is the main finding we wished to test in the replication to determine if the original findings were an anomaly or if something else was happening.

2.2 Previous Coursework

Recent previous work has identified problems with student understanding and learning in introductory courses. McCracken et al. [26] showed that students were unsuccessful at writing correct code to answer problems involving programming by the end of the introductory course. Lister et al. [22] took this investigation further by examining students' abilities to read and trace code. They concluded that weak students are often weak in their ability to read (and trace) code which is a precursor to writing code. In an attempt to better understand how reading and tracing are related to abilities to write code, Lopez et al. [23] created an instrument to determine a hierarchy of concepts (reading, tracing, solving Parsons problems, writing) with regard to code. Their work found that solving Parsons problems might actually be more difficult than tracing iterative code.

In an effort to prevent students from performing poorly, many have turned to trying to determine what best predicts computing proficiency, presumably as a way to determine how to better teach students and/or determine which student may be more likely to need extra support during their first course. The research on success predictors goes back decades, with numerous factors being identified such as comfort level [45], math and/or science background [5, 6, 20, 43, 45], spatial visualization skills [35], attributions of success and/or confidence [6, 34, 45], learning style [9, 35]. Watson [42] has argued that programming behavior was better predictor of success in the course than the traditional test-based prediction metrics and/or demographics.

Many believe that prior exposure to programming is a positive predictive factor in success in the first course. Hagan and Markam [17] showed that prior programming experience did in fact help students in the introductory course. They showed that the number of programming languages used was also important to students' initial success in the course. However, Evans and Simkin [16] showed that prior academic experience (including computing experience) did not factor strongly into a predictive model for success, but rather worked in concert with other behavior and cognitive factors to determine success. This was shown again by Wiedenbeck [44] in regards to non-majors taking a programming course and with Ventura [41] when looking at students taking an objects-first curriculum.

One thing from the predictors research that stands out is that there are likely a number of factors that contribute to a student's success in a course. Prior experience, whether it be in programming or other disciplines, is an important factor in some studies, but not as important in others. What we are interested in here is whether or not the prior experience impacts the understanding of a particular concept, not necessarily overall student success in the course.

3. METHOD OF STUDY

3.1 Purpose

Participants in introductory programming classes who had already been introduced to loops within their course were given additional instructional material designed to reinforce the practice of solving programming problems using `while` loops. Participants were recruited from 3 different first and second year programming courses at a technical university in the northeast United States and the study was conducted over a one month period.

Table 1 summarizes the differences between the three courses. The first two courses are first year, first semester courses serving primarily two different populations of students. The first course (101) serves as the first programming course (CS1 equivalent) for students intending to major in New Media Interactive Design (a College of Imaging Arts and Sciences major), or New Media Interactive Development (a College of Computing and Information Sciences major) and is taught using Processing [33]. The "New Media" majors are focused on the interaction of art and technology through media. The difference between the students is the focus of the major, the "design" major attracts primarily students who may consider themselves artists, while the "development" major attracts those who are more "technologists". The second course (105) serves as the first programming course (CS1 equivalent) for students intending to major in Game Design and Development (a College of Computing and Information Sciences major) and is taught in C# [33]. The game design and development degree is a technically focused degree in game design and development and the coursework has many similarities to a computer science degree. The department does not give credit for either 101 or 105 for Advanced Placement (AP) credit. Students earning high scores on the AP exam earn credit for another course from another department, but still need to take 101 or 105 to complete the requirements for their respective majors.

Table 1. Classes Participating in Study

Course	Programming Language	Majors	Experiment Delivery Method
101	Processing	New Media Interactive Design, New Media Interactive Development	Closed lab in-class exercise
105	C#	Game Design & Development	Optional at-home assignment
202	C# (some Processing)	New Media Interactive Design, Game Design and Development	Closed lab in-class exercise

The second year course (202) is designed to bring together the groups from New Media Interactive Development and Game Design and Development and is taught primarily in C# (with some limited time devoted to Processing) and focuses on the use and integration of media and media artifacts into interactive experiences [33]. It should be noted that students who take the 101 course take 2 more courses (102 and 201) before taking the follow-on course while those who take the 105 course take only 1 more course (106) before taking 202. So students from the 101 course path have a 3 semester sequence while those in the 105 track have a 2 semester sequence.

The study was conducted either in a closed lab setting with up to 30 computers in a single room, or as an optional at-home assignment (see Table 1). The participating instructors decided

how to structure the exercise in their particular course and what weighting it had on a student's grade, but participation in the study was strictly voluntary. That is, even in classes where there was a closed lab around the exercise, participation in the study described here was voluntary – consent to use the data for the study was given at the end of the exercises. All of these courses are taught in a computer lab of at most 30 students. Exercises where students are given a set of tasks to perform during the class period for their grade are a common part of these courses.

Students received an introduction to the study explaining that the material in the study was designed to help them learn how to write loops. Students were then given a URL to the first page of the study, which was housed in SurveyMonkey. Participants worked independently. The in-class sessions were an entire class period for the course (110 minutes). For the students who completed at home, the assignment was posted for them and they were given a due date by which they needed to complete the exercise. At the end of the window, the SurveyMonkey materials were closed.

3.2 Instructional Materials

The materials used were identical to those used in [28], other than placement of the consent. To learn the procedure for using `while` loops to solve programming problems, participants were given three worked examples and three practice problems. The worked examples and practice problems were interleaved so that after studying the first worked example, participants solved the first practice problem before moving on to the second worked example. The worked examples came in three formats, which varied between participants. The first format was not subgoal oriented, meaning that steps of the examples did not provide any information about the underlying subgoals of the procedure. 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., [24]). The third format grouped steps of the example by subgoal and provided a spot for participants to write generated subgoal labels for each group. Each of the groups was numbered as "label 1," "label 2," etc., and groups that represented the same subgoal had the same number; therefore, groups that represented subgoal 1 were numbered as "label 1" regardless of where in the example they appeared (see Figure 1). Participants were told that each of the worked examples would have the same subgoals, and they were encouraged to update and improve upon their generated labels as they learned more.

No labels	Given Labels	Placeholder for Label

sum = 0
lcv = 1
 	
WHILE lcv <= 100 DO
 	
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.

Participant groups also received different practice problems to test how contextual transfer may affect learning – the *isomorphic* transfer condition and the *contextual* transfer condition, just as in the original study. The contextual transfer was intended to be harder for participants to map concepts from the worked example to the practice problem. More difficult mapping can improve learning by reducing illusions of understanding caused by shallow processing thus inducing deeper processing of information [7, 15, 29]. However it can also increase cognitive load and potentially hinder learning [39].

After completing the instructions, participants completed novel programming assessments to measure their code writing performance. The assessments included two tasks. First, the code writing task asked participants to use the problem-solving structure that they had learned during the WE-PP pairs to solve four novel problems. Two of these problems required contextual transfer, meaning that they followed the same steps found in the instructions but in a different context. The other two problems required both contextual and structural transfer. In these problems the context was new to the participants and the solution to the problem required a different structure than the problems found in the instructional material (e.g., the practice problem is summing values, the assessment is counting matching values). These tasks were intended to measure participants' code writing and problem solving performance as a 'far' transfer.

After the code writing task, participants completed a Parsons problem. Parsons problems [30] involve correct code which is broken into code fragments that have to be put in the correct order. The Parsons problem used for assessment was a version of the "rainfall problem" [18]. 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.

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, code writing tasks, and time on task.

3.4 Participants

Participants were 100 students from a technical university in the northeast United States (Table 2). 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 computer, expected difficulty of the programming task, and primary language. There were no statistical differences between the groups for demographic data, which is expected because participants were randomly assigned to treatment groups.

Age	Gender	GPA	Major
M = 19	72% male	*M* = 3.5/4	33% New Media 63% Game Design 3% CS, SWE, CEngr

Table 2. Participant Demographics

Participants who did not attempt all tasks were excluded from analysis. For the replication piece of the study, 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. However, for the second piece of analysis within the paper, we looked at the success rate of all students who completed all the tasks. Based on these exclusion

criteria, we analyzed data from 27 participants for the replication study and 100 participants for overall performance.

3.5 Procedure

The procedure used in this study was identical to that used in [28], other than granting consent was moved from the beginning of the survey to the end. Students completed the demographic questions followed by a pre-test. The pre-test was comprised of multiple choice questions about `while` loops from previous Advanced Placement Computer Science A exams.

When participants finished the demographic questionnaire and pre-test, they began the instructional period which consisted of training followed by the worked example-practice problem (WE-PP) pairs. Participants who generated their own subgoal labels received training on how to create subgoal labels. Participants who did not generate their own subgoal labels received training to complete verbal analogies. Verbal analogies (e.g., water : thirst :: food : hunger) were considered a comparable task to subgoal label training because they both require analyzing text to determine an underlying structure. Participants who were not asked to generate their own labels were not given subgoal label training because it might have prompted them to process the instructions more similarly than would be expected of participants who were asked to generate their own labels, which might confound the results. The subgoal label training and the analogy training included expository instructions, worked examples, and activities.

Three WE-PP pairs were then presented to the participants. The worked example format differed between subjects among three levels: unlabeled, subgoal labels given, and subgoal labels generated. Furthermore, the transfer distance between worked example and practice problem differed between subjects between two levels: isomorphic or contextual transfer.

After the instructional period, participants moved to the assessment period. The assessment period included the problem solving tasks discussed earlier: 4 code writing problems and a Parsons problem.

At the end of the session, participants completed a post-test with the same questions as the pre-test to measure their learning. Throughout the procedure we recorded the time taken to complete each task. We also collected process data throughout the instructional period. We collected performance on the training activities and practice problems to ensure that participants were completing tasks.

We entered into the study with the following research questions:

R1. Do participants who learn with given subgoal labels and no contextual transfer perform better or worse on programming assessments than those who learn with given subgoal labels and contextual transfer?

R2. Do participants further along in their computing studies outperform novices on both the pre/post-test and programming assessments?

4. ANALYSIS AND RESULTS

4.1 Accuracy – Code Writing

We scored participants' solutions for accuracy to generate a code writing score. Participants earned one point for each correct line of code that they wrote. This scoring scheme allowed for more sensitivity than scoring solutions as wholly right or wrong. If participants wrote lines that were conceptually correct but contained typos or syntax errors (e.g., missing a parenthesis), they received points. We scored logic errors (having < rather an <=) as incorrect. We considered scoring for conceptual and logical accuracy more valuable than scoring for absolute syntactical accuracy because participants were still early in the learning process. Participants could earn a maximum score of 44.

In the statistics reported below, we include two types of effect sizes. The first, est. ω^2, describes how much of the variation in scores can be attributed to the manipulation. For example, for the code writing assessment, an est. ω^2 of .10 means that 10% of the variation in performance can be attributed to the instructional manipulations. The second, *f* or *d*, describes the difference between groups using the standard deviation as the unit of measurement. For example, for the code writing assessment, a *d* of .5 would mean that the difference- between the means of two groups is half of the standard deviation for those groups.

The effect of the interventions on code writing performance depended on the interaction of the worked example manipulation and transfer distance manipulation. We found no main effect of worked example format, F (2, 21) = 0.26, MSE = 105.6, p = .78, est. ω^2 = .02. In addition, we found no main effect of transfer distance, F (1, 21) = 1.47, MSE = 105.6, p = .24, est. ω^2 = .07. There was, however, a statistically significant interaction between worked example format and transfer distance, F (2, 21) = 5.19, MSE = 105.6, p = .015, est. ω^2 = .33, f = .44 (see Figure 2[1]).

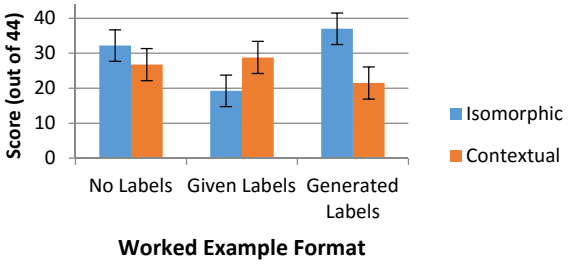

Figure 2. Code writing performance for novice programmers

In this interaction, the difference between the group that was given subgoal labels with isomorphic transfer (M = 18.4, SD = 13.0) and the group that was given subgoal labels with contextual transfer (M = 31.2, SD = 13.8) was not statistically significant, t (8) = -1.50, p = .17, but the difference between groups was large with an effect size of d = 0.95. Based on these results, the difference between groups is meaningful, even though it is not statistically significant, likely due to a small sample size. The sample size was small because this was a replication study thus we did not need as much statistical power to ensure the pattern of results was reliable. Because the effect size is large and matches previous results we conclude that this is a meaningful difference. Furthermore, the difference between the group that generated subgoal labels with isomorphic problems (M = 18.0, SD = 4.6) and the group that generated subgoal labels with contextual transfer (M = 37, SD = 4.5) was statistically significant, t (4) = 7.18, p = .002, with a large effect size, d = 4.18. This effect size is based on a sample of six, therefore, it is likely not reliable. The effect size is likely inflated due to the small sample; however, the effect of the intervention is still valid and in the correct direction. These results mean that participants who were given subgoal

[1] Error bars on all bar graphs represent the 95% confidence interval.

labels performed better when they had contextual transfer, and participants who generated subgoal labels performed better with isomorphic problems.

Performance on the post-test was similar to that on the pre-test. Average scores on the post-test were low, 36% (1.8 out of 5 points). We found no statistical differences for main effect of worked example format, F (2, 21) = .07, MSE = 1.21, p = .93, est. ω^2 = .01, main effect of transfer distance, F (1, 21) = .28, MSE = 1.21, p = .60, est. ω^2 = .01, or interaction, F (2, 21) = .93, MSE = 1.21, p = .41, est. ω^2 = .09.

4.2 Accuracy – Parsons Problem

We scored participants' Parsons problem answers for correct order to create their score. Participants ranked the 13 code pieces from the Parsons problem; 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 because they are following the correct piece. This scoring scheme better captures participants' understanding than scoring for absolute correct order as it does not penalize correct sequences of code that follow incorrect sequences.

Based on the analysis of Parsons problem responses, performance did not seem to depend on the worked example or transfer manipulations. We found no statistical differences for main effect of worked example format, F (2, 21) = 1.20, MSE = 9.70, p = .32, est. ω^2 = .10, main effect of transfer distance, F (1, 21) = .61, MSE = 9.70, p = .45, est. ω^2 = .03, or interaction, F (2, 21) = .24, MSE = 9.70, p = .79, est. ω^2 = .02.

4.3 Time Efficiency

4.3.1 Time on Worked Example-Practice Pairs

For time spent studying worked examples and solving practice problems, we found a meaningful difference based on worked example format even though there was not a statistically significant ANOVA, F (2, 21) = 3.37, MSE = 202.2, p = .06, est. ω^2 = .27, f = .35. We found no main effect of transfer distance, F (1, 21) = 1.94, MSE = 202.2, p = .18, est. ω^2 = .10. In addition, we found no interaction, F (2, 21) = 1.80, MSE = 202.2, p = .19, est. ω^2 = .17 (see Figure 3).

4.3.2 Time on Code Writing Assessments

Time spent on the code writing assessments was not affected by the subgoal or transfer manipulations. We found no statistical differences for a main effect of worked example format, F (2, 21) = 2.22, MSE = 49.9, p = .13, est. ω^2 = .17, main effect of transfer distance, F (1, 21) = .63, MSE = 49.9, p = .44, est. ω^2 = .03, or interaction, F (2, 21) = .53, MSE = 49.9, p = .60, est. ω^2 = .05.

4.3.3 Time on Parsons Problems

Time spent on the Parsons problem was not affected by the subgoal or transfer manipulations. We found no statistical differences for main effect of worked example format, F (2, 21) = 1.14, MSE = 8.75, p = .34, est. ω^2 = .10, main effect of transfer distance, F (1, 21) = .21, MSE = 8.75, p = .65, est. ω^2 = .01, or interaction, F (2, 21) = .26, MSE = 8.75, p = .77, est. ω^2 = .02.

5. PREVIOUS COURSEWORK EXPERIENCE

As mentioned earlier, we also asked students in a follow-on programming course to participate in the study. This section reviews their performance and compares it with the novice performance. For this analysis we looked at all students who completed the tasks regardless of their pre-test score.

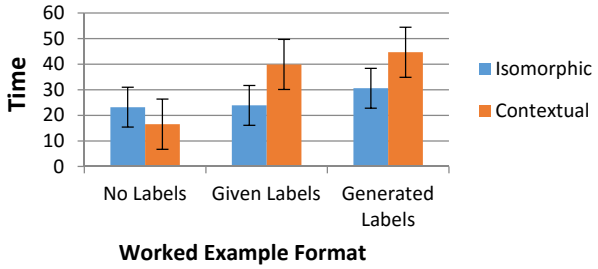

Figure 3. Time (in minutes) on instructional tasks

5.1 Results of All Introductory Students

Students considered for the results in Section 4, the replication study, had to match the qualifications of the original study. This meant that we excluded all introductory students that correctly answered 3 or more questions on the pre-test correctly. This eliminated 24 students, almost as many as we analyzed (27). It became clear that many of the students in the introductory courses had significant loop writing knowledge prior to our intervention. This led us to further investigate if this prior knowledge could be attributed to prior coursework.

Looking at all participants in the introductory courses (n = 51), the average score on the pre-test was 46% (2.3 out of 5). Participants scored about the same on the post-test with an average of 54% (2.7 out of 5).

Within this group, no manipulation by itself made a statistical difference in code writing performance. There was no main effect of worked example format, F (2, 45) = .32, MSE = 106.1, p = .73, est. ω^2 = .01. There was also no main effect of transfer distance, F (1, 45) = 1.88, MSE = 106.1, p = .18, est. ω^2 = .04. There was, however, an interaction, F (2, 45) = 4.04, MSE = 106.1, p = .024, est. ω^2 = .15 (see Figure 4). This interaction resembles the pattern of results seen in the analyses reported earlier in this paper.

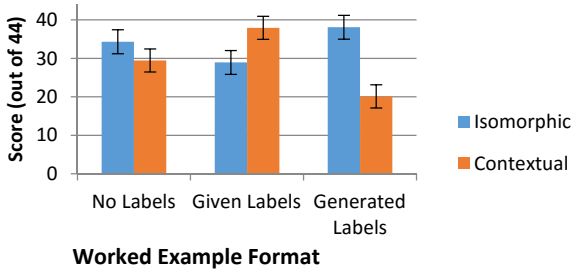

Figure 4. Code writing performance for participants in introductory courses

On the Parsons problem, there were no statistical differences among groups on performance. There was no main effect of worked example format, F (2, 45) = 1.69, MSE = 12.6, p = .19, est. ω^2 = .07. There was no main effect of transfer distance, F (1, 45) = .75, MSE = 12.6, p = .39, est. ω^2 = .02. There was no interaction, F (2, 45) = .02, MSE = 12.6, p = .98, est. ω^2 = .001.

5.2 Results of Students in Follow-On Course

Participants in the follow-on course (i.e., the course after the

introductory college course) were excluded from previous analyses because they had at least a semester of programming instruction, and the instruction in the study was designed for novices. Participants in this course scored on average a 73% (3.6 out of 5) on the pre-test. After instruction on the post-test, participants scored about the same with an average of 70% (3.5 out of 5). Within this group, participants who were given subgoal labels performed better on the code writing assessments than those who were not given labels or those who generated labels, F (2, 43) = 7.33, MSE = 15.9, p = .002, est. ω^2 = .25, f = .37. There was no main effect of transfer distance, F (1, 43) = .25, MSE = 15.9, p = .62, est. ω^2 = .01, nor was there an interaction, F (2, 43) = 2.30, MSE = 15.9, p = .11, est. ω^2 = .10 (see Figure 5).

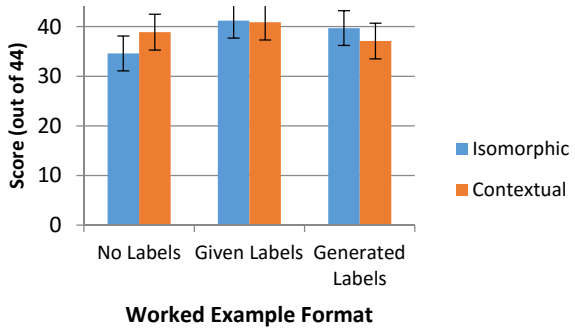

Figure 5. Code writing performance for 202 students

On the Parsons problem, participants who generated subgoal labels performed worse than those who were not given labels or those who received labels, F (2, 43) = 4.75, MSE = 10.9, p = .014, est. ω^2 = .18, f = .31. There was no main effect of transfer distance, F (1, 43) = 2.17, MSE = 10.9, p = .15, est. ω^2 = .05, nor was there an interaction, F (2, 43) = .62, MSE = 10.9, p = .54, est. ω^2 = .03 (see Figure 6).

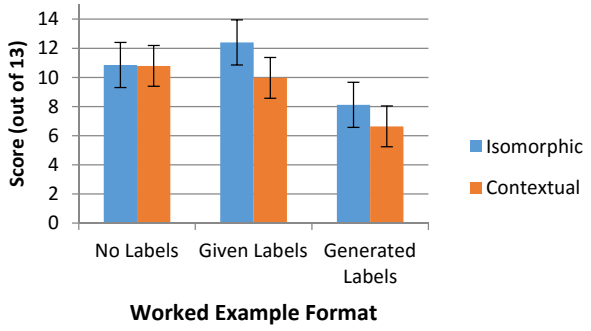

Figure 6. Parsons problem performance for 202 students

5.3 Effect of Previous Coursework

To explore the effect of computing courses in high school and the effect of prior computing courses in college on performance, we used these two variables as random independent variables in ANOVA to determine if they affected performance. Participants who took computing courses in high school performed better on the code writing assessment than those who did not, F (1, 96) = 12.0, MSE = 56.2, p = .001, est. ω^2 = .11, f = .35. Participants who had taken prior computing courses in college performed better, F (2, 96) = 14.3, MSE = 56.2, p < .001, est. ω^2 = .13, f = .38. There was an interaction, F (2, 96) = 11.1, MSE = 56.2, p = .001, est. ω^2 = .10, f = .33, such that participants who took computing courses in high school did not perform better that those who did not in later college computing courses (Figure 7). In other words, it does not matter if the previous course was taken in high school or college – the fact that the student had a previous course predicts better performance; but that advantage does not continue into the next course.

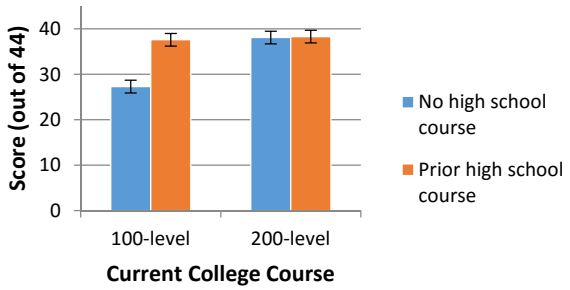

Figure 7. Code writing performance by course level

Similar to the code writing task, participants who took computing courses in high school performed better on the Parsons problem than those who did not, F (1, 96) = 9.85, MSE = 11.6, p = .002, est. ω^2 = .09, f = .31. Participants who had taken prior computing courses in college also performed better, F (2, 96) = 6.78, MSE = 11.6, p = .011, est. ω^2 = .07, f = .26. For this assessment, however, there was no interaction, F (2, 96) = 1.66, MSE = 11.6, p = .20, est. ω^2 = .02, suggesting that those who had computing courses in high school performed better than those who did not, even after their first computing course in college.

6. DISCUSSION

Here we summarize our findings related to our original research questions and discuss the implications for computing education.

6.1 Replication

Our replication of [28] yielded results that support the original findings. This study confirms that novice participants who learn by generating subgoal labels (using isomorphic WE-PP pairs) perform the best, and statistically better than if they had been WE-PP pairs with contextual transfer (Figure 2). We conclude that for the best learning results novice students should be taught to generate their own subgoal labels but be given WE-PPs that are very similar.

We hypothesize that teaching novice students to generate their own subgoal labels does require additional time, both for instruction and for the student during the WE-PP instruction time. Additionally it should be noted that within this experiment participants did not receive any feedback on the appropriateness of their generated labels. To obtain maximum benefit from generating subgoal labels, students should receive feedback on the correctness of their labels. Alternately, similar learning results may be obtained by using given subgoal labels.

However, if pre-defined given subgoal labels are used, the WE-PP pairs should utilize contextual transfer to ensure maximum learning. As mentioned earlier, this is contradictory to what would be predicted by CLT. This is certainly one phenomenon that needs further research. It may be that with given subgoal labels and isomorphic problems students do not adequately self-explain the process associated with each subgoal as the steps are identical within both the worked example and practice problem. Just as in the first study, we reviewed student code submissions to ensure

that they were not copied from the worked example and they were not. Also the time spent in the instructional period indicates that participants spent similar amounts of time regardless if they received isomorphic or contextual transfer WE-PP pairs.

It may be that with given subgoal labels students require multiple examples for comparison to be able to determine the generalities of the labels and the process. It would be interesting to determine if more examples or additional practice problems would improve the learning performance of these groups. It may also be possible to present the worked example with no contextual story at all – just as a simple problem to be solved. If students are presented with a "vanilla" worked example with given subgoal labels followed by a practice problem embedded within a context, would the performance differ? Looking into these and other possibilities are planned as future research areas.

6.2 Previous Coursework

We were surprised at the number of students in the introductory courses that were excluded due to their pre-test scores. In looking at the demographics, we noticed many of the introductory students had a previous computing course in high school and their pre-test scores reflected this prior experience. In examining their performance along with the students in the follow-on course, we found that having that prior coursework experience made a significant difference in the performance on the assessment tasks.

Participants in the introductory courses with high school coursework experience performed statistically better on both the code writing and Parsons problem assessments than students in the same courses without high school computing coursework. These students performed similarly to those in the follow-on course (202). Students in the 202 course performed statistically better on the Parsons problem assessments than the students in the introductory courses, regardless of high school coursework. We have evidence that students with some coursework experience, whether in high school or a previous college class, have actually mastered this concept given the pre-test score above 70% and the performance on the code writing assessment (Figure 5 for 202 students only).

However, as can be seen in Figure 7, students in the follow-on course that did not have a high school computing course performed approximately the same as those who did have a high school computing course on the code writing assessment task. Thus, those without the previous coursework advantage had "caught up" to those who started with more knowledge. This has significant implications for those teaching introductory programming. If students with prior programming experience are in the same class as those with previous computing coursework, we cannot expect them to perform the same on assessment tasks, especially after a short exposure. The participants in the introductory courses had been exposed to loops in their current courses for approximately 2 weeks and were preparing for a graded test which would include loops. Even without knowing the scores for the exam, we predict that the students with previous computing courses in high school would significantly outperform those without that experience.

As instructors we need to be cognizant of the potential influence that prior coursework may have on student performance. We should not expect those being exposed to concepts for a minimal amount of time to be as proficient as those who have had a semester's worth (or more) of practice. Perhaps students with prior experience should be in a separate class from those without prior coursework, even for introductory courses. The evidence from this study suggests that by the next course, any advantage those with prior coursework in high school once had, has been eliminated; at least for code writing tasks.

On the Parsons problem assessment, results from this study indicate that those who took a high school course continue to outperform those who did not. Further research is needed to determine if this result is repeatable and why this might be the case. It may be that continuous code writing practice improves performance on Parsons problems. We are unaware of research that compares novice to expert performance on Parsons problems but this study suggests that additional practice in programming continues to improve Parsons problem performance.

We found one additional interesting result in this study. Students in the follow-on course who generated subgoal labels performed statistically worse than those in the same course who were not given labels or those who received labels. This may be an example of the expertise reversal effect [19]. The expertise reversal effect occurs when the learner is presented with information that causes them to think below their automatized schema. The instructional design material, in this case the generation of subgoal labels, uses working memory that would not have been necessary if the learner were just solving the problem. In other words, the participant could have solved the problem without any instructional material at all because of their prior knowledge. The instructional material interfered with their problem solving process. Further research into when subgoal labels should no longer be used with those learning programming should be explored.

7. CONCLUSION

This study was originally conceived as a means to replicate an existing study to determine if the puzzling results would be confirmed. The data gathered in this study confirms that students who learn with given subgoal labels perform better with contextual transfer between the WE-PP pair than those who received isomorphic WE-PP pairs. While we still have no evidence as to why this occurs, contrary to cognitive load theory, we now know that the result is repeatable and deserves further research to investigate why this group in computing differs from those in other disciplines.

Because of the number of participants eliminated from the replication analysis we explored reasons for the difference in performance. Students who had previous computing coursework outperformed those without previous coursework in both code writing and Parsons problem assessment tasks. Students with previous computing coursework should be assessed at a different standard than those with minimal time exposure to the concept. However, in the follow-on course this performance difference disappears indicating that those without previous coursework do "catch up". This indicates that the students with differing experience backgrounds can be merged into a single class.

While some may think the results of this paper, students with previous experience perform better, are obvious, we demonstrate that our students actually do eventually learn and master a concept (writing loops) unlike so much previous research [22, 26, 36]. We also provide evidence that any advantage gained through previous coursework disappears, with regard to this introductory concept, in the follow-on course. We find these facts, that our students actually do learn and that students without previous experience can catch up, very encouraging.

8. ACKNOWLEDGMENTS

We would like to thank the students who participated in the study and their instructors who graciously gave us the time. We also thank the anonymous reviewers who supplied comments which improved this paper.

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.

9. 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. *Fourth International Conference of the Learning Sciences* (2000).

[5] Bennedsen, J. and Caspersen, M.E. 2005. An investigation of potential success factors for an introductory model-driven programming course. *Proceedings of the first international workshop on Computing education research* (2005), 155–163.

[6] Bergin, S. and Reilly, R. 2005. Programming: factors that influence success. *ACM SIGCSE Bulletin* (2005), 411–415.

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

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

[9] Campbell, V. and Johnstone, M. 2010. The significance of learning style with respect to achievement in first year programming students. *Software Engineering Conference (ASWEC), 2010 21st Australian* (2010), 165–170.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] Evans, G.E. and Simkin, M.G. 1989. What best predicts computer proficiency? *Communications of the ACM*. 32, 11 (1989), 1322–1327.

[17] Hagan, D. and Markham, S. 2000. Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin* (2000), 25–28.

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

[19] Kalyuga, S. 2007. Expertise reversal effect and its implications for learner-tailored instruction. *Educational Psychology Review*. 19, 4 (2007), 509–539.

[20] Leeper, R.R. and Silver, J.L. 1982. Predicting success in a first programming course. *ACM SIGCSE Bulletin*. 14, 1 (1982), 147–150.

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

[22] Lister, R. et al. 2004. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin* (2004), 119–150.

[23] Lopez, M. et al. 2008. Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the fourth international workshop on computing education research* (2008), 101–112.

[24] 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.

[25] 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.

[26] McCracken, M. et al. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working group reports from ITiCSE on Innovation and technology in computer science education* (Canterbury, UK, 2001), 125–180.

[27] 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.

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

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

[30] 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.

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

[32] 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.
[33] Rochester Institute of Technology 2014. Undergraduate Course Descriptions.
[34] Rountree, N. et al. 2004. Interacting factors that predict success and failure in a CS1 course. *ACM SIGCSE Bulletin* (2004), 101–104.
[35] Simon et al. 2006. Predictors of success in a first programming course. *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (2006), 189–196.
[36] Soloway, E. and Ehrlich, K. 1984. Empirical studies of programming knowledge. *Software Engineering, IEEE Transactions on*. 5 (1984), 595–609.
[37] Sweller, J. et al. 1998. Cognitive architecture and instructional design. *Educational psychology review*. 10, 3 (1998), 251–296.
[38] Sweller, J. et al. 2011. *Cognitive load theory*. Springer.
[39] Sweller, J. 2010. Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational psychology review*. 22, 2 (2010), 123–138.
[40] van Gog, Tamara and Paas, Fred 2012. Cognitive Load Measurement. *Encyclopedia of the Sciences of Learning*. Springer.
[41] Ventura Jr, P.R. 2005. Identifying predictors of success for an objects-first CS1. (2005).
[42] Watson, C. et al. 2014. No tests required: comparing traditional and dynamic predictors of programming success. *Proceedings of the 45th ACM technical symposium on Computer science education* (2014), 469–474.
[43] White, G. and Sivitanides, M. 2003. An empirical investigation of the relationship between success in mathematics and visual programming courses. *Journal of Information Systems Education*. 14, 4 (2003), 409.
[44] Wiedenbeck, S. 2005. Factors affecting the success of non-majors in learning to program. *Proceedings of the first international workshop on Computing education research* (2005), 13–24.
[45] Wilson, B.C. and Shrock, S. 2001. Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin* (2001), 184–188.