2016

Employing Subgoals in Computer Programming Education

Lauren Margulieux
Georgia State University, lmargulieux@gsu.edu

Richard Catrambone
Georgia Institute of Technology, richard.catrambone@psych.gatech.edu

Mark Guzdial
Georgia Institute of Technology, guzdial@cc.gatech.edu

Recommended Citation

Margulieux, Lauren; Catrambone, Richard; and Guzdial, Mark, "Employing Subgoals in Computer Programming Education" (2016). *Learning Technologies Division Faculty Publications*. 3.
https://scholarworks.gsu.edu/ltd_facpub/3

Employing Subgoals in Computer Programming Education

Lauren E. Margulieux[a], Richard Catrambone[a], and Mark Guzdial[b]

[a]*School of Psychology, Georgia Institute of Technology, Atlanta, USA*

[b]*School of Interactive Computing, Georgia Institute of Technology, Atlanta, USA*

Lauren E. Margulieux (corresponding author), School of Psychology, Georgia Institute of Technology, Atlanta, GA, USA 30332-0170, l.marg@gatech.edu, 1-214-538-9769; Richard Catrambone, School of Psychology, Georgia Institute of Technology, Atlanta, GA, USA 30332-0170, rc7@prism.gatech.edu; Mark Guzdial, School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA, USA 30332-0760, guzdial@cc.gatech.edu.

This research was supported by a grant from Georgia Institute of Technology's GVU Center and Institute for People and Technology. The authors would like to thank Barbara Ericson for her consultation and Frank Durso for critiques on earlier versions of the manuscript. The authors would also like to thank Joe Dagosta, Catherine Hwang, Hannah Fletcher, and Andrea Crews.

Correspondence concerning this article should be addressed to Lauren Margulieux, School of Psychology, Georgia Institute of Technology, Atlanta, GA 30332-0170. Email: l.marg@gatech.edu

Employing Subgoals in Computer Science Education

The rapid integration of technology into our professional and personal lives has left many education systems ill-equipped to deal with the influx of people seeking computing education. To improve computing education, we are applying techniques that have been developed for other procedural fields. The present study applied such a technique, subgoal labeled worked examples, to explore whether it would improve programming instruction. The first two experiments, conducted in a laboratory, suggest that the intervention improves undergraduate learners' problem solving performance and affects how learners approach problem solving. A third experiment demonstrates that the intervention has similar, and perhaps stronger, effects in an online learning environment with in-service K-12 teachers who want to become qualified to teach computing courses. By implementing this subgoal intervention as a tool for educators to teach themselves and their students, education systems could improve computing education and better prepare learners for an increasingly technical world.

Keywords: subgoal learning; online learning; computing education; worked examples; professional development.

Introduction

As information technology becomes ubiquitous, technical skills are becoming increasingly important for individuals to be effective in many aspects of their lives (Scaffidi, Shaw, & Myers, 2005). Individuals want to understand the computing technologies in their lives, and to construct computing technologies that meet their needs. This desire to understand and construct computing technologies has caused a rapid increase in demand for computing education that has left many education systems struggling to adapt. This struggle in the education system is two-fold. First, techniques for teaching computing are relatively underdeveloped compared to some other domains, such as physics. Best practice in computing education has been developed by copying practices from industry or other disciplines (Porter, Guzdial, McDowell, & Simon, 2013). For this reason, relatively few computer science (CS) teaching methods are developed with rationale from educational psychology (Pears et al., 2007).

Second, the number of educators qualified to teach computing is insufficient. Few states in the US track the number of CS teachers, but advanced placement (AP) CS is offered nationwide. Because few high schools have more than one AP CS teacher, the number of schools offering AP CS is our best indicator of CS teachers (Wilson, Sudol, Stephenson, & Stehlik, 2010). Today, there are only about 2300 teachers in the United States who teach high school AP Computer Science, for the approximately 25,000 to 30,000 high schools in the US. A major national effort is underway to raise the number of high school CS teachers to 10,000 so that at least one out of every three high schools in the US has a teacher capable of teaching CS (Astrachan, Cuny, Stephenson, & Wilson, 2011). Both underdeveloped teaching methods and the lack of teachers can be addressed by developing methods to improve computing education and using those methods to prepare more high school CS teachers.

As a domain that focuses on problem solving, CS is similar to other procedural domains, like mathematics, in the way that procedures are taught. Like students in many other procedural domains, students who are learning to program have difficulty conceptually understanding the procedures that they are taught and rely heavily on analogies to worked examples (Linn & Clancy, 1992). Just as Chi, Feltovich, and Glaser (1981) found that novice physics students tend to focus on the contextual features of problems (whether an example used a ramp) instead of the structural features (which law applies to the problem), Anderson, Farrell, and Sauers (1984) found that novice programming students make the same types of mistakes. Moreover, as is the case in many procedural domains, the cognitive load of novice programming students tends to be high (Lister, 2011). To address these issues in programming education, an educational method used in other procedural domains was adapted for programming.

Aiding students to learn the subgoals for solving problems has helped them focus on the structure of a procedure and reduced cognitive load in procedural domains with relatively little cost (in terms of both development and instruction time; Catrambone, 1996, 1998). Subgoals are components of complex problem solutions that provide functional pieces of the final solution. The individual steps taken to achieve a subgoal can vary, but the subgoals needed to solve problems within a subset of a domain typically do not (Catrambone, 1994). By learning the subgoals and how to achieve them, a student can use that knowledge to solve novel problems more effectively (Catrambone, 1994).

To encourage subgoal learning, subgoal labeled worked examples have been used. Worked examples are popular in procedural domains because procedural text (i.e., general instructions about the procedure) tends to be too abstract to easily grasp, and students learn as much or more through specific examples as through general descriptions of procedures (LeFevre & Dixon, 1986). Additionally, worked examples provide a solution for a learner to study that they can use to guide independent problem solving (Atkinson, Derry, Renkl, & Wortham, 2000). Worked examples, however, can be misleading for novices because novices do not recognize what information in a worked example is important to the problem solving procedure (i.e., structural information) and what information is irrelevant to the procedure (i.e., superficial information; e.g., Chi et al., 1981). Highlighting information in worked examples is a common scaffolding technique to help learners identify important information in the examples and use them more efficiently for learning (e.g., Yan & Lavigne, 2014).

To highlight structural information, subgoal labeled worked examples group steps into underlying subgoals, and these groups are labeled with the subgoal that they achieve. For example, a subgoal in a probability problem might be to represent the number of possible outcomes; therefore, all of the steps that are involved in determining possible outcomes would be grouped under the label “Find possible outcomes.” By providing this extra information, subgoal labels help students learn the subgoals that form the problem solving procedure, which improves their problem solving performance (Catrambone, 1994). Consistently, people who receive subgoal labeled worked examples are more likely to solve novel problems successfully than people who receive the same worked examples without the subgoal labels (e.g., Catrambone, 1998).

Theories of self-explanation and cognitive load can explain why subgoal labels are effective. Cognitive load theory posits that humans balance three types of cognitive load vying for our cognitive resources during learning: intrinsic (associated with the information necessary to complete a task, such as a formula), extraneous (associated with how the information is being taught as well as unnecessary contextual features, such as specific details of a worked example), and germane (associated with processing information in a way that will create long-term learning, such as self-explanations; Sweller, 2010). Because cognitive resources are limited, if intrinsic or extraneous load is too high, resources will not be available for germane load (Sweller, 2010). While worked examples can be effective in instruction, they can introduce extraneous cognitive load because they require the learner to attend to information specific to the example that is not strictly related to the procedure being taught. Intrinsic cognitive load is already high in programming instruction because students are learning problem solving procedures at the same time that they are learning to use the programming language (Trafton & Reiser, 1993). This situation would be similar to learning to solve math problems at the same time as learning to read and write numbers. Therefore, when students use worked examples while learning to program, few cognitive resources are left over to devote to germane cognitive load.

Subgoal labels reduce extraneous cognitive load caused by worked examples by emphasizing information that is important to understanding examples and helping students focus on pertinent information. Catrambone (1994) found that when worked examples include subgoal labels, the labels help the learner identify and focus on the structural information in the example. Furthermore, subgoal labels highlight the underlying structure of the examples and visually chunk problem solving steps, allowing learners to easily identify these functional chunks. With this format of examples, learners can more easily compare components of a solution between and within examples (Catrambone, 1996). These types of comparisons promote self-explanation (e.g., by prompting students to question why two groups of steps are both characterized by the same subgoal label; Bielaczyc, Pirolli, & Brown, 1995; Renkl & Atkinson, 2002). Moreover, promoting self-explanation externally through the structure of an example (e.g., using subgoal labels; Atkinson et al., 2000) has a consistent, positive impact on the quantity and quality of self-explanations (Catrambone, 1996; Renkl & Atkinson, 2002).

Research has found that worked examples that prompt students to self-explain are highly successful for improving learning (e.g., Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Hilbert & Renkl, 2009), but not all prompts successfully promote self-explanation. Catrambone (1998) found that subgoal labels that provide meaningful information about the purpose of a set of steps were more successful for novices than for advanced learners. On the other hand, abstract subgoal labels that required the learner to create their own explanation of the purpose of steps was more successful for advanced learners than for novices. This pattern of results is consistent with the expert reversal effect of cognitive load theory in that the novices needed more support and scaffolding to most successfully solve problems and the advanced learners needed less support to most successfully solve problems (Sweller, 2010). Because this study focuses on novices, meaningful subgoal labels will be used to promote self-explanation.

Overview of Experiments

The present study compared the effectiveness of subgoal labeled worked examples to that of conventional worked examples (i.e., without subgoal labels) for teaching novices to use Android App Inventor. It was hypothesized that learners who received the subgoal labeled examples would perform better while solving novel programming problems than learners who received the conventional examples. The first experiment tested these two formats of examples in a laboratory setting to examine whether subgoal labeled worked examples would improve problem solving performance. The second experiment added a talk-aloud component to the laboratory experiment to potentially provide converging evidence for subgoal formation due to different formats of examples. Finally, the third experiment replicated the results from the laboratory experiments in a more ecological learning environment with learners who were interested in learning the programming procedure.

App Inventor is used to create applications (apps) for Android devices, and it was chosen for the study because it was selected by the majority of K-12 teachers polled as the language that they most wanted to learn. It is also a drag-and-drop programming language that is free to use, so it is highly accessible. Drag-and-drop programming languages replace writing code with dragging components from a menu. The App Inventor interface can be explored at http://appinventor.mit.edu/. This type of code creation can be easy to understand and removes syntax as a possible source of confusion for learners (Hundhausen, Farley, & Brown, 2009).

App Inventor has two interfaces. The first is the Designer through which the user makes components for apps. For instance, if an app included a button to be clicked, the button would have been created in the Designer (see Figure 2). The second is the Blocks Editor through which the user programs behaviors for the components of the app using pieces of program called blocks. For instance, if the app played a sound when a button was clicked, the programming for this behavior would have been created in the Blocks Editor (see Figure 3).

Material Development

Demographic data about learners were collected including information about their age, gender, academic field of study, SAT scores, high school and college GPA, year in school, number of complete college credits, prior CS courses taken, primary language, number of math courses completed, comfort with computers, and expected difficulty of learning a programming language. These demographics are possible predictors of performance (Rountree, Rountree, Robins, & Hannah, 2004).

Worked examples were adapted from the projects section of a distance education website for teaching programming languages (Ericson, 2012). The present study used both video demonstrations of the task (e.g., a video of someone creating the app) and textual instructions for how to complete the task (see Figure 4 for a sample). The video demonstrations showed participants the procedure to create each of the apps; therefore, they served as the worked examples. Due to the amount of time it takes to complete an app, each session taught participants to create one app. Because the worked examples were necessarily long, all instructions regardless of condition were split into manageable chunks. The video for each session was two to three minutes long. The textual instructions for each app were between 49 and 65 steps long and visually split into chunks no longer than 18 steps.

Video demonstrations were used because multimedia presentations of procedures allow learners to process more information at once because they use both visual and auditory systems (Mayer, 2009). Because participants needed to be taught to use the App Inventor interface as well as the problem solving procedure, using a video to present the interface visually and the procedure verbally was considered appropriate. Textual instructions were also used because Palmiter and Elkerton (1993) found that though video demonstrations can quickly and naturally show users how to learn a direct-manipulation interface, participants in their study who watched a video retained less of that information one week later than participants who read text instructions. Participants in the present study were asked to make the apps shown as examples because theories of knowledge compilation and scaffolding suggest that studying worked examples *and* applying procedures leads to better learning than studying alone (Pea, 2004; Trafton & Reiser, 1993). For these reasons, the present study used both video, to introduce how to make the app, and text, to guide participants through making the app. Prior research suggests that presenting subgoal labels throughout the examples is effective because it shows the structure of the example (Catrambone, 1998). In the subgoal condition, subgoal labels were incorporated throughout both the video demonstration and the textual instruction. The subgoal labels were determined using the Task Analysis by Problem Solving (TAPS) approach (Catrambone et al. 2014) with subject-matter experts (see Figure 1 for the list of subgoals).

Experiment 1

Participants learned to make apps in App Inventor from worked examples, either with or without subgoal labels, and then solved novel programming problems. The sessions were completed in a computer-based learning environment with an experimenter present to give administrative instructions about the experiment and answer administrative questions.

Method

Participants

Participants were 40 students recruited from a psychology participation pool of a medium-sized, urban university. They received course credit for their participation. To participate in the experiment, students must have been at least 18 years of age, and they must not have taken more than one computer programming or CS class. Experience with Android App Inventor disqualified potential participants to ensure that all participants had the same level of prior experience.

Procedure

The experiment had two one-hour sessions that were one week apart. A computer-based learning environment was used to provide all instruction about App Inventor. In each session, participants received procedural instruction to help them make an app (a different app in each session) including the video demonstration and textual instructions. After completing the instructions, participants completed a series of novel tasks that measured problem solving performance. Participants had a limited amount of time to complete these tasks, similar to an exam, but the time restrictions allowed ample time based on pilot testing results. In addition to solving tasks at the end of each session (i.e., immediate assessment), participants solved tasks at the beginning of the second session after a week-long delay (i.e., delayed assessment; see Table 1 for the format of sessions). During the delayed assessment, participants had access to the App Inventor interface but nothing from the previous session. In the sessions, experimenters answered questions about the study (e.g., “Can I watch the video again?”) or provided technical support (e.g., reopen the video if participants closed it) but did not answer questions about the instructions or App Inventor (e.g., “How do I make a button?”).

In the first session, participants completed a demographic questionnaire before starting the instructions. Next, participants had 40 minutes to use the instructional video and text to make the Music Maker app that plays a sound when an object on the screen is touched or the device is tilted. Finally, they had 15 minutes to complete the first immediate assessment, which had five problem solving tasks.

In the second session, participants had 10 minutes to complete the delayed assessment, which had four problem solving tasks. Next, participants had 25 minutes to use the video and text to make the Fortune Teller app that randomly selects and displays a fortune from a list of fortunes when a button is pressed. Finally, they had 25 minutes to complete the last assessment, which had five problem solving tasks.

Measurements

The problem solving tasks asked participants to list the steps that they would take to add or edit features of apps (see Figure 5 for the tasks). Participants were not allowed to refer to the video or text during the problem solving tasks, but they were allowed to use the App Inventor interface and the app that they had created in that session. Typically, participants completed as much of the solutions as they could in the interface and then wrote the steps that they took. Asking participants to write down steps, instead of directly scoring their solutions in the interface, allowed participants to gain extra points by describing steps that they would take even if they could not find the correct menu in the interface to execute it. Because participants had no experience with this interface prior to the experiment, scoring for conceptual accuracy rather than absolute accuracy seemed more appropriate.

A participant did not have to successfully complete a task in order to move on to the next task. This independence was important because problem solving tasks were intended to be difficult (e.g., average score intended to be about 50% correct) to avoid restricting the range of the data. Tasks were classified as either component or procedural transfer tasks (see Figure 6). To solve *component* transfer tasks, participants followed the same procedure that they had used during the instructional period, and they substituted blocks or components of the *same* type. For example, one component transfer task asked participants to program the clap sound to play when the phone was tilted up. To complete this task, participants could follow the same steps that were in the instructions to program the drum sound to play when the phone was tilted to the right. They simply had to replace the drum sound with the clap sound and the x-axis acceleration sensor with the y-axis acceleration sensor.

To solve *procedural* transfer tasks, participants had to achieve the same subgoals that they had used during the instructional period, but they had to use blocks or components of a *different* type. That is, they were solving problems in the same topic, but they could not follow the same procedure that the video or text showed. For example, one task asked participants to program an ImageSprite to move five pixels to the right when touched. The steps to do this task were different than the steps used during the instructional period, but the subgoals that needed to be achieved, “handle events in My Blocks” and “set output in My Blocks,” were the same.

Because procedural transfer tasks required using new components of the interface, participants were told which type of blocks they should use to complete the task. This guidance provided the declarative information needed for the task but did not provide information about how to use the block. For example, participants received a task that required them to use blocks that they had not used during the instructional period related to the orientation sensor. To alert participants to the component needed to complete the task, they received the hint: “You’ll need to make an orientation sensor.” Because the assessments were timed, the hints might have decreased error variance by reducing the time participants spent searching for suitable features.

To score problem solving performance, the correct solutions for the tasks were deconstructed into the functional parts necessary to complete the solution (e.g., creating an output for the app). Because the solutions for the assessment tasks are complex, scoring the pieces of each solution instead of scoring the entire solution as correct or incorrect allowed for more sensitivity (see Figure 7).

Problem solving performance was primarily represented by two scores: a “correct” score (i.e., number of parts a participant completed correctly) and an “attempted” score (i.e., number of parts a participant attempted). Completing a part correctly was operationally defined as listing all of the steps required to complete that part. Attempting a part was operationally defined as listing at least one of the steps required to complete the part, listing an incorrect step that would achieve a similar function, or describing the purpose of the part in some way. The attempted score was meant to measure whether participants could identify the parts of the solution that are needed regardless of their success at producing a correct solution.

Participants’ responses were scored by two raters. Raters were undergraduate research assistants trained by scoring examples with an experimenter’s supervision until proficiency was reached, which was after about five examples for each problem solving task. Given the highly procedural nature of the tasks, few errors or disagreements occurred in scoring.

Design

The independent variable was the type of worked example (subgoal labeled worked examples, n = 20, or conventional worked examples, n = 20); thus there were two groups in the experiment. The dependent measures were performance on the problem solving tasks (number of parts of the solution completed correctly and number of parts attempted) and time to complete each assessment. Participants were randomly assigned to conditions.

Results and Discussion

Demographic information was collected as possible predictors of performance. Random assignment of participants to groups was used so that demographic differences between groups were not expected. None of the demographics had statistically significant ($p < .05$) correlations with performance except that expected difficulty of learning a programming language correlated positively with amount of time spent on assessment phases, $r = .38$, $p = .02$. There was not, however, a significant difference between groups on this variable, t (38) = .48, std. error = .41, $p = .63$. This finding means that participants who expected learning a programming language to be more difficult tended to spent more time on the assessments, but both groups were approximately equal on expected difficulty. Therefore, this correlation is not expected to confound the performance results. This finding suggests that the results presented below were not significantly impacted by the demographics collected, and their impact will not be discussed further. Table 2 has mean demographic data for both groups.

Across the 14 assessment-task solutions, there were a total of 46 parts; therefore, participants could get a maximum score of 46 for both the correct and attempted measures. Inter-rater reliability for the measures was high with a one-way random model intraclass correlation coefficient of agreement (ICC(A)) of .97. The effect size, f, is given for the following analyses. It is the index used for analysis of variance.

Problem Solving Performance

Across all assessments, two immediate and one delayed, participants in the subgoal group completed 36% more of the problem solving tasks correctly (M = 28.1, 61%; SD = 7.2) than the conventional group (*M* = 20.6, 45%; *SD* = 6.7), *F* (1, 38) = 11.16, *MSE* = 48.71, *p* = .002, est. ω^2 = .23, *f* = .53. This result means that participants who received subgoal labeled worked examples solved novel problems better (i.e., defined as achieving more correct parts of the solution) than participants who received conventional worked examples.

For both component and procedural transfer tasks, the subgoal group performed better than the conventional group suggesting that subgoal labels improve both types of transfer (see Table 3). Because the *component* transfer tasks were very similar to the examples shown in the video and text, better performance on these tasks suggests that the subgoal labels helped participants remember or reconstruct the instructed procedures better. Better performance on the *procedural* transfer tasks suggests that the subgoal labels helped participants to adapt and generalize the procedures to novel problems. Because the differences in scores on component and procedural transfer between groups were equivalent, the following analyses used a combined score to avoid loss of power.

To examine the results more closely, each of the three assessments were analyzed independently. During the immediate assessment at the end of Session 1 (maximum score of 15), participants in the subgoal group completed 44% more of the problem solving tasks correctly (*M* = 8.2, 55%; *SD* = 3.6) than the conventional group (*M* = 5.7, 38%; *SD* = 3.1), *F* (1, 38) = 5.49, *MSE* = 11.22, *p* = .025, est. ω^2 = .13, *f* = .37. During the immediate assessment at the end of Session 2 (maximum score of 20), participants in the subgoal group completed 28% more of the problem solving tasks correctly (*M* = 14.0, 70%; *SD* = 3.8) than the conventional group (*M* = 10.9 (55%), *SD* = 3.4), *F* (1, 38) = 6.86, *MSE* = 13.26, *p* = .013, est. ω^2 = .16, *f* = .41.

In the delayed assessment at the beginning of Session 2 (maximum score of 11), participants had access to the App Inventor website but not to anything from the previous session. On the delayed assessment, subgoal participants completed 46% more of the problem solving tasks correctly (*M* = 6.0, 55%; *SD* = 2.6) than conventional participants (*M* = 4.1, 37%; *SD* = 1.8), *F* (1, 38) = 7.06, *MSE* = 4.97, *p* = .01, est. ω^2 = .16, *f* = .42.

To explore the pattern of scores across the assessments, a repeated measures analysis was used. The interaction of assessment and group was not significant suggesting that the subgoal group consistently outperformed the conventional group at about the same level across assessments (see Figure 8), Λ_{Pillai} = .002, *F* (2, 36) = .03, *p* = .97.

As an illustration of the impact of subgoal labels, consider their effect on participants' likelihood of defining a variable. One of the problem solving tasks asked participants to create a list, which was similar to the list that they created while making the Fortune Teller app in Session 2. An important part of completing this task is defining the variable that contains the list because without defining the variable, other parts of the program would not be able to reference the list. Having difficulty with referencing part of a program is a common problem with programming students (e.g., for object-oriented systems; Guzdial, 1995).

Though all participants correctly defined the variable during the instructional period (i.e., while they had access to the worked example), participants in the subgoal group were five times more likely to define the variable in this problem solving task (*M* = .6, *SD* = .5) than the conventional group (*M* = .1, *SD* = .2), *F* (1, 38) = 15.12, *MSE* = .16, *p* < .001, est. ω^2 = .29, *f* = .61. This finding could mean that the label helped the subgoal participants to recognize the underlying structure of the example, and this recognition helped them transfer what they learned while solving the task. However, this result could also mean that the subgoal label "define variable" simply increased the subgoal participants' exposure to the idea of defining a variable, and that the extra exposure helped them to remember those steps. These possibilities were explored in Experiment 2 with a talk aloud procedure.

Attempted Tasks and Time on Task

Participants in the subgoal group attempted to complete 18% more of the problem solving tasks (*M* = 34.7, 75%; *SD* = 6.1) than the conventional group (*M* = 29.4, 64%; *SD* = 7.4), *F* (1, 38) = 5.91, *MSE* = 45.91, *p* = .02, est. ω^2 = .14, *f* = .38. This finding could mean that participants in the subgoal group could better identify the parts of the solution that are needed regardless of whether they could complete the solution correctly. By attempting more parts, the subgoal participants could be demonstrating that they had a higher-level view of the problem state than the conventional participants. This possibility was explored more in Experiment 2.

The subgoal group finished the assessment phases 11% faster (*M* = 40.6 min., *SD* = 7.5 min.) than the conventional group (*M* = 45.5 min., *SD* = 5.1 min.), *F* (1, 38) = 5.48, *MSE* = 41.07, *p* = .03, est. ω^2 = .13, *f* = .37. The correlation between time and number of correct parts was nonsignificant, *r* = .06, *p* > .05; therefore, there is not a linear relationship between time on task and problem solving performance.

Summary of Results

In Experiment 1, for every measure of problem solving performance, participants who received subgoal labels performed better than participants who did not. The extra instruction provided by the subgoal labels significantly improved problem solving even though the labels themselves are only a few words. This difference in performance could be due to lower extraneous cognitive load that left more mental resources for learning. Additionally, subgoal labels could have helped learners understand the structure of examples, leading to better transfer. Subgoal labels can also be helpful because they can help learners create effective mental representations of problem solving procedures. To better understand the difference between groups, Experiment 2 examined participants' problem solving processes.

Experiment 2

This follow-up experiment further explored the effect of subgoal labels on learning a programming procedure. This experiment focused on capturing differences in participants' problem solving processes to help explain differences in problem solving performance between groups. To capture the problem solving process and illuminate ways that the groups differed, a talk-aloud protocol and extra measures were used.

Method

Participants

Participants were 12 students recruited from a medium-sized, urban university who received course credit for their participation. The sample size for this experiment was small because of the time demands for collecting and analyzing data for the talk-aloud protocol. Having six participants per condition is considered adequate for collecting qualitative data (e.g., Nielsen, 1999). Criteria for participation were the same as for Experiment 1.

Procedure

The procedure for Experiment 2 was identical to Experiment 1 except that while participants completed the problem solving tasks, they engaged in a talk-aloud protocol. Participants were asked to explain their goals or strategies for completing the tasks while solving the tasks. Before starting the first assessment, participants practiced the talk-aloud procedure by playing a game of tic-tac-toe with the experimenter who gave them feedback about articulating their process (e.g., articulating their perception of the problem solving state or their strategies for completing the problem).

Design

The independent variable for Experiment 2 was the same as for Experiment 1: subgoal labels (*n* = 6) or no labels in the examples (*n* = 6). In addition to qualitative analysis of the talk-aloud data, another dependent measure was added to examine how participants solved problem tasks. This measure helped evaluate the process of problem solving in addition to the end product. Participants were scored on the number of blocks (i.e., the pieces of code used to make a program) that they dragged out while solving tasks. This measure addressed how efficiently participants solved problem tasks because a low number of blocks would indicate that they used only the blocks that were needed and a high number of blocks would indicate that they used unnecessary blocks, for example, to explore incorrect solutions. Number of blocks was compared to performance to ensure that a low number of blocks did not indicate low effort on the task. Time taken to complete assessments was not measured because of the variability caused by the talk-aloud protocol.

Results and Discussion

Means for demographic data can be found in Table 2. None of the collected demographics correlated with performance, and the data will not be discussed further. Participants' comments were qualitatively analyzed to determine the types of information that participants communicated. Before data were collected, experimenters were interested in comments similar to the "exploring" and "planning" categories described below. After collecting and reviewing data, however, these categories were deemed too general, and the four other categories (and the miscellaneous category) were added for the final analysis. Comments from the talk-aloud transcripts were then scored for whether they described the following seven types of information (all comments were categorized into these seven types):

- Explaining – describing the purpose of their actions (e.g., "This will tell it to do something when the button is pushed,")
- Exploring – experimenting with different combinations of blocks without a specific goal (e.g., "Maybe these two blocks go together,")
- Gathering – looking up information in "Help" or searching through menus for a specific block with a specific goal (e.g., "I'm looking for a block that lets me set the x coordinates,")
- Evaluating – determining if their solutions were correct (e.g., "I think that's right, but I'm looking for something else that might work,")
- Planning – describing what part of the task to do next or what needs to be done (e.g., "Next, I need to set the output,")
- Inferring – making generalizations about how App Inventor or the procedure works (e.g., "Oh, so images must be put on the canvas to be activated by touch.")
- Miscellaneous – comments not related to the task (e.g., "It's cold in here.")

Inter-rater reliability was acceptable with an ICC(A) of .86. Comments in the miscellaneous category were uncommon, and they were not included in the analyses.

Results of this analysis can be found in Table 4. Participants who received subgoals made more than twice as many comments (total = 187) as participants who did not (total = 76). For this reason, participants' comments will be discussed by both the number of comments and the percentage of comments. For example, based on number of comments, participants in the subgoal group made more than twice as many comments to explain their actions as in the conventional group (80 versus 33, respectively); based on percentage of comments, however, 42% of both groups' comments were explanations. The ratios are similar for comments about gathering information. Though the reason for this difference in volume is unclear, it is possible that subgoal participants had a lower cognitive load while solving problems and thus were better able to articulate their process compared to conventional participants.

Participants who received subgoal labels made more comments about planning and evaluating, both in raw numbers and as a percentage of total comments. Both planning and evaluating are valuable metacognitive strategies for problem solving (Livingston, 2003). Because subgoal participants made more comments about these processes, the subgoal labels might have prompted participants to engage in metacognition by highlighting the structure of examples in the instruction. Subgoal participants were also the only participants to make inference comments which used information from a specific problem to inductively understand the general procedure. This finding supports the argument that participants in the subgoal group were better at transferring knowledge.

Participants were also given a score for the number of blocks (i.e., pieces of code) that they dragged out while solving tasks. Each block that was dragged from the menu to the programming area counted as one point; moving blocks within the programming area (including putting them in the trash can) did not affect this score. Participants in the subgoal group dragged out 62% fewer blocks while working through tasks (*M* = 18.8, *SD* = 13.1) than those in the conventional group (*M* = 49.9, *SD* = 5.7), *F* (1, 10) = 9.84, *MSE* = 148.14, *p* = .02, est. ω^2 = .62, *f* = .91. Furthermore, number of blocks dragged out during a task was negatively correlated with performance on that task, *r* = -.32, *p* = .032, suggesting that using fewer blocks was related to better performance. These findings suggest that participants who received subgoal labels were more likely to have a more efficient strategy during problem solving than those who did not.

Problem Solving Performance

The performance metrics used in Experiment 1 were also collected in Experiment 2. The sample size of Experiment 2 precludes strong conclusions to be made from these metrics, but the results for this experiment are included as evidence that the results of Experiment 1 were replicated. Participants could get a maximum score of 46 for both correct and attempted parts. Due to the small number of participants in this experiment (*n* = 12), there was little power, thus the correct and attempted measures were analyzed by effect size between groups and not analyzed separately for type of transfer or assessment. The effect size for correct parts was *f* = .59, and the effect size for attempted parts was *f* = .42. These effect sizes are similar to those found in Experiment 1 (*f* = .53 and *f* = .38, respectively) suggesting that if the sample size matched that of Experiment 1, the same statistically significant differences would have been observed.

Besides replicating the effect sizes from the first experiment, these results show that without time constraints, subgoal participants again performed better than conventional participants. This finding is important because it means that participants who received subgoals outperformed those who did not even though all participants had as much time as they wanted to work on the problems.

The results of this experiment suggest that participants who received subgoal labels were more effective problem solvers. When participants from both groups used planning strategies for solving the tasks, they would often state the part that they were going to work on and then fill out the steps to complete it. This pattern suggests that they identified a high-level function that they wanted to complete and then filled in the low-level details. This structural approach is similar to how experts solve problems (Anderson et al., 1984). The difference between groups was that subgoal participants used this strategy more often than conventional participants. Subgoal labels give learners additional information about the problem solving process by breaking the procedure into functional steps. This extra information can influence how learners mentally represent the procedure (Catrambone, 1998) resulting in more structurally-focused problem solving.

Four participants (all in the subgoal label group) said, unprompted at the end of the experiment, that they thought the sessions were interesting and that they had fun. Instruction and practice in programming can often be frustrating for students, and motivating students is difficult (Kinnunen & Simon, 2011). Finding a combination of conditions that makes computing education fun and engaging for the majority of students is one of the goals in programming instruction (Clancy & Linn, 1990).

Experiment 3

Experiments 1 and 2 included face-to-face contact with an experimenter in a controlled environment. The aim of Experiment 3 was to examine whether the same subgoal label manipulation would improve problem solving in an online learning setting that had neither of these features. Furthermore, previous experiments used samples of college students whereas this experiment tested the manipulation with a sample from the K-12 teacher population to inform the creation of professional development for teachers who want to become qualified to teach computing courses. The protocol was expanded from two sessions to four sessions to make the training more substantial in an effort to make it more appealing to potential teacher participants. The third and fourth sessions, like the second session in prior experiments, included a delayed assessment, instruction for a new app, and an immediate assessment.

Method

Participants

Participants were middle and high school, in-service teachers recruited through mailing lists for teachers interested in computer science education (e.g., via local chapters of the Computer Science Teachers Association). These teachers self-identify as computer science teachers. Participants were disqualified if they had prior experience with Android App Inventor but were not restricted by other prior experience. The teachers' self-reported backgrounds varied on a number of factors such as education, years as a teacher, years teaching CS, and professional development for computing education. Participants mean age was 48.4 (*SD* = 10.75) and 48% of them were men.

Completion rates for the sessions tapered throughout the study with 27 people starting the first session, 18 completing the second session, and 10 people completing all four sessions. This attrition is comparable to that found in other online learning environments (Ho et al., 2015). Though the teachers volunteered to participate in the study, they were not compensated in any way other than receiving free instruction. The problem solving tasks were intended to be difficult in order to avoid a restriction of range problem caused by all participants performing well. Many of the teachers made comments indicating that they were frustrated by the difficulty level of the assessments. The most common complaint was that the instructions did not provide specific information about how to solve the assessment tasks (particularly for procedural transfer tasks). Participants might have lost motivation to complete the sessions due to lack of compensation. In addition, some teachers had unforeseeable conflicts arise that ended their participation. There is not a recognizable pattern that differentiates participants who completed the study from participants who did not based on either their assigned condition or demographic data. Due to low completion rates, data from only the first two sessions were analyzed (i.e., *n* = 18). Though a larger sample size would have been ideal, this experiment was designed as an attempt to replicate previous results in a new instructional context with a different population. A small sample size can be sufficient for that purpose.

Procedure

The experiment was conducted entirely online. Instructions and media for the apps were emailed to participants, and the sessions were hosted on SurveyMonkey (http://surveymonkey.com). Each SurveyMonkey survey gave participants instructions for viewing the video and text and for completing the problem solving tasks. Through the survey, participants were asked to record how long they spent using the instructional video and text to make the app and how long they spent working on each problem solving task. Participants were also asked to rate the difficulty of making each app and completing each task on a Likert-type scale from “1-Very Difficult” to “7-Very Easy.”

Sessions were emailed one week apart to participants. The first and second sessions taught the same apps as in Experiments 1 and 2. The additional third and fourth sessions taught participants to make an app that counted the number of times the user pressed a button in a set time frame and taught them to make an app similar to the game Pong. The timestamp on the surveys were checked to ensure participants completed the sessions no fewer than six days and no more than eight days apart. To reduce the demand on teachers’ time, the number of problem solving tasks was reduced. The type of tasks in Experiment 3 was the same as in Experiments 1 and 2, but there were fewer of them in each assessment. Participants were explicitly instructed to not look at instructions during the assessments.

Design

The design for Experiment 3 was the same as Experiment 1 except that the amount of time participants spent creating each app and working on assessments was not limited.

Participants' subjective ratings of the difficulty to make the app during instruction and complete the problem solving tasks were also recorded.

Results and Discussion

There were no statistically significant correlations between participant performance and prior experience at p = .1. The alpha was made more stringent (in this case .1 because a null result is sought) due to the small sample size. In the problem solving tasks for this experiment, there were 32 parts; therefore, participants could get a maximum score of 32 for both the correct and attempted measurements. Inter-rater reliability was high with a one-way random model intraclass correlation coefficient of agreement (ICC(A)) of .87.

Problem Solving Performance

Participants in the subgoal group (n = 9) completed 81% more of the problem solving tasks correctly (M = 26.6, 83%; SD = 5.1) than the conventional group (n = 9, M = 14.7, 46%; SD = 6.6), F (1, 16) = 18.23, MSE = 34.89, p = .001, est. ω^2 = .53, f = 1.01. Because the experiment was conducted online, there is no way to know whether participants referred to the worked examples during the assessments. The first two experiments ensured that participants not reference worked examples during the assessments, but in this online experiment, we could only ask participants' cooperation. The worked examples that participants received were the same except for the subgoal labels. If participants used the worked examples to help them solve the tasks, then the effect size would be expected to decrease if problem solving performance was affected by anything other than subgoal labels. The fact that the effect size for problem solving performance was larger than it was in Experiment 1 supports the conclusion that the subgoal labels affected performance.

To examine these general findings more closely, each of the three assessments were analyzed independently. During the first immediate assessment (maximum score of 11), participants in the subgoal group completed more than two times the problem solving tasks correctly (*M* = 9.7, 88%; *SD* = 1.4) than the conventional group (*M* = 3.0, 27%; *SD* = 3.0), *F* (1, 16) = 27.04, *MSE* = 5.56, *p* < .001, est. ω^2 = .63, *f* = 1.23. During the second immediate assessment (maximum score of 10), participants in the subgoal group completed 70% more of the problem solving tasks correctly (*M* = 8.0, 80%; *SD* = 2.8) than the conventional group (*M* = 4.7, 47%; *SD* = 3.6), *F* (1, 16) = 4.82, *MSE* = 10.38, *p* = .043, est. ω^2 = .23, *f* = .50. Both results suggest that the subgoal group was better at solving novel problems than the conventional group, but the effect size of the first assessment phase was larger than that of the second assessment phase (*f* = 1.23 vs. *f* = .50, respectively).

This finding has only two data points, but it could be the case that the difference between groups decreases with repeated exposure to the same type of material. Though further research would be necessary to explore this possible limitation, this decrease could be expected because as novices gain more knowledge, they are better able to identify important information and need less external guidance. Subgoal labels' role is to highlight the important information for novices who are not able to recognize it for themselves.

All of the tasks in the delayed assessment for Experiment 3 were component transfer tasks (maximum score of 11). For this assessment, participants in the subgoal group completed 48% more of the problem solving tasks correctly (*M* = 9.0, 82%; *SD* = 1.7) than the conventional group (*M* = 6.1, 55%; *SD* = 3.2), *F* (1, 16) = 6.17, *MSE* = 6.41, *p* = .024, est. ω^2 = .27, *f* = .57. This result suggests that the subgoal intervention can lead to better retention of problem solving procedures.

Time on Task and Perceived Difficulty

There were no statistically reliable differences between the groups on the time and difficulty measures (see Table 5). These results suggest that participants in the subgoal group performed better than the conventional group without necessarily taking longer to complete the instructions or tasks and without feeling as though they had to work harder. These findings are important because learners are motivated to make the learning process as easy as possible (Eiriksdottir & Catrambone, 2011), and our manipulation could improve problem solving performance without increasing the perceived difficulty of learning.

Overall, the results of Experiment 3 are consistent with the results of Experiments 1 and 2, but with even larger effect sizes. These results suggest that a subgoal intervention could be effective in an online learning environment, such as a K-12 teacher professional development program for computer programming. This finding has encouraging practical implications for the effectiveness of the subgoal intervention in learning environments with no instructor as well as implications for being distributed widely at low cost.

Conclusion

The results of the present study suggest that subgoal labeled worked examples can improve programming performance. The results can be applied to improve programming instruction for both students and future computing teachers. The results also add to the literature on subgoal and STEM learning by demonstrating that the subgoal intervention can be effective in a new domain (i.e., programming), new environment (i.e., online), and for a new population (i.e., K-12 in-service teachers).

The present study informs our understanding of the generalizability of subgoal labels as an instructional technique. Previously, when subgoal labels have been tested in worked examples, the learner was presented with the multiple instances of the subgoal labels through several, short examples. In this study, the learner was presented with multiple instances of the subgoal labels through one, long example. This distinction is important because exposure to multiple examples generally increases learners' capability to transfer problem solving knowledge (Atkinson et al., 2000; Bassok, 1990). The results of the present study suggest that exposure to multiple instances of a subgoal label, regardless of the number of examples, might be the key to subgoal labels' effect, which can save a considerable amount of time when problem solving procedures and examples are complex, such as in programming education.

Limitations and Future Directions

The sample sizes for Experiments 2 and 3 were relatively small. Given that these experiments were largely attempting to replicate findings from Experiment 1, a larger sample size would not be expected to provide more information. Because of the labor-intensive nature of coding data in Experiment 2 and sampling pool used in Experiment 3, these sample sizes seem appropriate. Similarly, the attrition rate in Experiment 3 was not ideal though not unexpected given the online learning environment and lack of compensation. Because a difference was not found between participants who completed the study and those who did not, the main limitation from attrition in this case is the small sample size.

In Experiment 3, as learners gained more experience with App Inventor, the difference between groups shrunk. This result is based on two data points, however, and the same pattern of results was not found in Experiment 1 or 2. Expertise reversal effect (e.g., Sweller, 2010) would predict that some forms of instruction are less effective as students develop expertise. We predict that subgoals might still be useful to students with more expertise, but at a higher level of abstraction. Subgoal labels for students with greater expertise might represent more steps than subgoal labels for students with less expertise. For example, an algebra student learning to solve for a variable might use lower level subgoals (e.g., put matching terms on one side of the equation) to complete the task, but a calculus student learning to solve a derivative might think of solving for a variable as a subgoal. This capacity to scale to the knowledge level of the learner allows subgoal labels to be useful past the novice stage of learning and is an opportunity to explore in future research.

Considering results across experiments, the difference between groups found in Experiment 3 (i.e., with K-12 teachers) was about twice as large as the difference between groups found in Experiment 1 (i.e., with college students; *f* = 1.01 vs. *f* = .53, respectively), even though it was conducted in a less controlled environment and with participants of more varied backgrounds. Though we did not find many effects of individual differences within each experiment, this disparity suggests that there might be an overall population difference between the college students and the teachers. The population differences, however, are confounded by methodological differences in delivery method for the sessions. Based on research about the differences between face-to-face and online delivery methods for instruction (Bernard et al., 2004), a difference in performance due to these different modes of delivery would not be expected, but more research would be needed to test these expectations.

Comparing participants' scores directly across the experiments is not possible because the assessments had a different number of tasks (though the tasks were the same). However, if participants' scores from Experiment 1 are recalculated to include only the problem solving tasks that were in Experiment 3, then the conventional group in Experiment 1 (*M* = 14.0, *SD* = 5.8) scored about the same as the conventional group in Experiment 3 (*M* = 14.7, *SD* = 6.6). However, the subgoal group in Experiment 1 (*M* = 19.9, *SD* = 5.6) scored lower than the subgoal group in Experiment 3 (*M* = 26.6, *SD* = 5.1). Using the recalculated scores from Experiment 1, the effect size in Experiment 1 changes little (*f* = .56 compared to *f* = .53).

One explanation for this difference in subgoal labels on problem solving performance could be level of motivation. The teachers volunteered because they wanted to learn the content to further their careers, whereas the undergraduates volunteered to earn class credit and were likely less motivated to learn the content. Motivation might increase the effect of subgoal labels on performance. Another possible explanation concerns prior knowledge; the teachers on average had more experience with computer science than the undergraduates. The subgoal labels might have helped the teachers to identify underlying structures and make analogies to prior computer science knowledge, thus improving their problem solving.

The present study found that subgoal labels are effective in an online learning environment with no face-to-face interaction. Because the intervention is built into the instructions that learners receive, the increase in learners' performance is not dependent on an instructor. The intervention, therefore, could be effective in an array of learning environments including those without an instructor. Even with an instructor, the subgoal labels could be helpful because they prompt the instructor to convey subgoal information and provide a tool for accomplishing this.

Our society needs more people to learn computer science to fully utilize technological advances and perpetuate these advances. Thus, research on improving computer science instruction is particularly timely. The present study demonstrates the effectiveness of the subgoal intervention for programming instruction that could potentially be widely implemented relatively easily and inexpensively for a variety of learners.

References:

Anderson, J. Farrell, R. & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 3*(2), 87-129. doi: 10.1207/s15516709cog0802_1

Astrachan, O., Cuny, J., Stephenson, C., & Wilson, C. (2011). The CS10K project: mobilizing the community to transform high school computing. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (SIGCSE '11). ACM, New York, NY, USA, 85-86. doi:10.1145/1953163.1953193

Atkinson, R.K., Derry, S.J., Renkl, A., & Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. *Review of the Educational Research, 70*(2), 181-214. doi: 10.3102/00346543070002181

Bassok, M. (1990). Transfer of domain-specific problem-solving procedures. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *16*(3), 522-533. doi: 10.1037/0278-7393.16.3.522

Bernard, R.M., Abrami, P.C., Lou, Y., Borokhovski, E., Wade, A., Wozney, L., Wallet, P.A., et al. (2004). How does distance education compare with classroom instruction? A meta-analysis of the empirical literature. *Review of Educational Research, 74* (3), 379-439.

Bielaczyc, K., Pirolli, P., & Brown, A.L. (1995). Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. *Cognition and Instruction, 13,* 221-252.

Catrambone, R. (1994). Improving examples to improve transfer to novel problems. *Memory and Cognition, 22*, 605-615.

Catrambone, R. (1996). Generalizing solution procedures learned from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 22*, 1020-1031. doi: 10.1037/0278-7393.22.4.1020

Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General, 127,* 355-376. doi: 10.1037/0096-3445.127.4.355

Catrambone, R., Gane, B.D., Adams, A.E., Bujak, K.R., Kline, K.A., & Eiriksdottir, E. (2014). Task analysis by problem solving (TAPS): A method for uncovering expert knowledge. *Under review*.

Chi, M.T.H., Feltovich, P.J., Glaser, R. (1981). Categorization and representations of physics problems by experts and novices. *Cognitive Science*, *5,* 121-152.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science, 13*, 145-182.

Clancy, M., & Linn, M. (1990). Functional fun. *SIGCSE Bulletin*, *22*(1), pp 63-67. Doi: 10.1145/323410.319085

Ericson, B. (2012, February 12). ICE Distance Education Portal. Retrieved from http://ice.cc.gatech.edu/dl/?q=node/641

Guzdial, M. (1995). Centralized mindset: a student problem with object-oriented programming. In *Proceedings of the 26th SIGCSE TechnicalSsymposium on Computer Science Education,* Curt M. White, James E. Miller, and Judy Gersting (Eds.). ACM, New York, NY, USA, 182-185. doi: 10.1145/199688.199772

Hilbert, T.S., & Renkl, A. (2009). Learning how to use a computer-based concept-mapping tool: Self-explaining examples helps. *Computers in Human Behavior, 25*(2), 267-274.

Ho, A. D., Chuang, I., Reich, J., Coleman, C. A., Whitehill, J., Northcutt, C. G., Williams, J. J., Hansen, J. D., Lopez, G., & Petersen, R. (2015). HarvardX and MITx: Two years of open online courses Fall 2012 – Summer 2014. Retrieved from http://papers.ssrn.com/sol3/ papers.cfm?abstract_id=2586847

Hundhausen, C.D., Farley, S.F., & Brown, J.L. (2009). Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study. *ACM Transactions in CHI, 16*(3). doi: 10.1145/1592440. 1592442

LeFevre, J.. & Dixon, P. (1986). Do written instructions need examples? *Cognition and Instruction, 3*, l-30. doi: 10.1207/s1532690xci0301_1

Linn, M.C., & Clancy, M.J. (1992). The case for case studies of programming problems. *Commununications of the ACM, 35* (3), 121-132. doi: 10.1145/131295.131301 http://doi.acm.org/10.1145/131295.131301

Lister, R. (2011). Computing education research: Programming, syntax and cognitive load. *ACM Inroads, 2* (2), 21-22. doi:10.1145/1963533.1963539

Livingston, J. (2003). Metacognition: An overview. Retrieved from http://files.eric.ed.gov/fulltext/ED474273.pdf

Mayer, R.E. (2009). Multimedia learning (2nd). New York: Cambridge University Press.

Nielsen, J. (1999). Designing web usability: The practice of simplicity. New Riders.

Palmiter, S., & Elkerton, J. (1993). Animated demonstrations for learning procedural computer-based tasks. *Human-Computer Interaction, 8*(3), 193-216. doi:10.1207/ s15327051hci0803_1

Pea, R.D. (2004). The social and technological dimensions of scaffolding and related theoretical concepts for learning, education, and human activity. Journal of the Learning Sciences, 13(3), 423-451. doi: 10.1207/s15327809jls1303_6

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *SIGCSE Bulletin, 39*(4), 204-223. doi:10.1145/1345375.1345441 http://doi.acm.org/10.1145/1345375.1345441

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works?. *Communications of the ACM, 56*(8), 34-36. doi:10.1145/2492007.2492020

Renkl, A., & Atkinson, R.K. (2002). Learning from examples: Fostering self-explanations in computer-based learning environments. *Interactive Learning Environments, 10*(2), 105-199.

Rountree, N., Rountree, J., Robins, A., & Hannah, R. (2004). Interacting factors that predict success and failure in a CSI course. *SIGCSE Bulletin, 33*(4), pp 101-104. doi: 10.1145/1044550.1041669

Scaffidi, C., Shaw, M., & Myers, B. (2005). An approach for categorizing end user programmers to guide software engineering research. In *Proceedings of the first workshop on End-user software engineering* (pp. 1-5). ACM, New York, NY. doi: 10.1145/1082983.1083232

Sweller, J. (2010). Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational Psychology Review, 22*(2), 123-138. doi: 10.1007/s10648-010-9128-5

Trafton, J.G., & Reiser, B.J. (1993). The contributions of studying examples and solving problems to skill acquisition. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society* (pp. 1017-1022). Boulder, CO.

Wilson, C., Sudol, L.A., Stephenson, C., & Stehlik, M. (2010). *Running on Empty*. New York, NY: ACM and CSTA.

Yan, J. & Lavigne, N.C. (2014). Promoting college students' problem understanding using schema-emphasizing worked examples. *Journal of Experimental Education, 82*(1), 74-102.

Table 1

Order of Experimental Sessions

Session	1st section	2nd section	3rd section
1	Introduction	Instruction	Assessment
2	Assessment	Instruction	Assessment

Table 2

Means for Demographic Data Collected Separated by Condition

Condition	Gender	*Age*	*SAT*	*College GPA*	*Year in College*	*Prior CS course*	Comfort with computer	Expected difficulty
				Experiment 1				
Subgoal	61% M	20.1	1251	3.46	2.4	39% Y	5.75	3.85
Conv	61% M	19.7	1282	3.36	2.0	44% Y	5.85	3.65
				Experiment 2				
M subgoal	67% M	20.7	1266	3.12	3.13	50% Y	6.00	2.67
M conv	57% M	20.7	1293	2.94	2.91	67% Y	6.12	3.14

Note: Gender reported in percent of participants who were men. SAT scores out of 1600. Prior CS courses reported in percentage of participants who had taken a CS course before. Comfort with computers and expected difficulty of learning a programming language were measured on a 7-pt. scale (1-Not comfortable at all; Very difficult and 7- Very comfortable; Very easy).

Table 3

Difference between Conditions for Component and Procedural Transfer Tasks in Experiment 1

Transfer	*M* subgoal	*M* conv	$\overline{SD}$	*F*	*p*	est. ω^2	*f*
Component	12.6	8.3	3.6	13.66	.001	.27	.58
Procedural	8.1	5.8	3.1	5.45	.025	.13	.37

Table 4

Number and Percentage of Comments Made between Groups for Each Type of Information

Category	Subgoal Number	Subgoal Percentage	Conventional Number	Conventional Percentage
Explaining	80	43%	33	43%
Exploring	18	10%	19	25%
Gathering	18	10%	5	7%
Evaluating	26	14%	7	9%
Planning	42	22%	12	16%
Inferring	3	2%	0	0%

Note: Time in minutes and difficulty on 7-pt. scale (1-Very Difficult and 7-Very Easy).

Table 5

Difference between Conditions for Time and Difficulty Measures in Experiment 3

Category	*M* subgoal	*M* conv	$\overline{SD}$	*F*	*p*
Time on Instruction	77.3	87.8	37.8	.37	.55
Difficulty of Instruction	4.9	4.5	1.0	.23	.64
Time on Assessment	76.6	56.7	33.1	1.44	.25
Difficulty of Assessment	4.3	3.8	1.1	.66	.43

Note: Time in minutes and difficulty on 7-pt. scale (1-Very Difficult and 7-Very Easy).

1. Create component
2. Set properties
3. Handle events in My Blocks
4. Set output in My Blocks
5. Set conditions from Built-In
6. Define variable in Built-In
7. Emulate App

Figure 1. List of subgoal labels used in subgoal labeled worked examples.

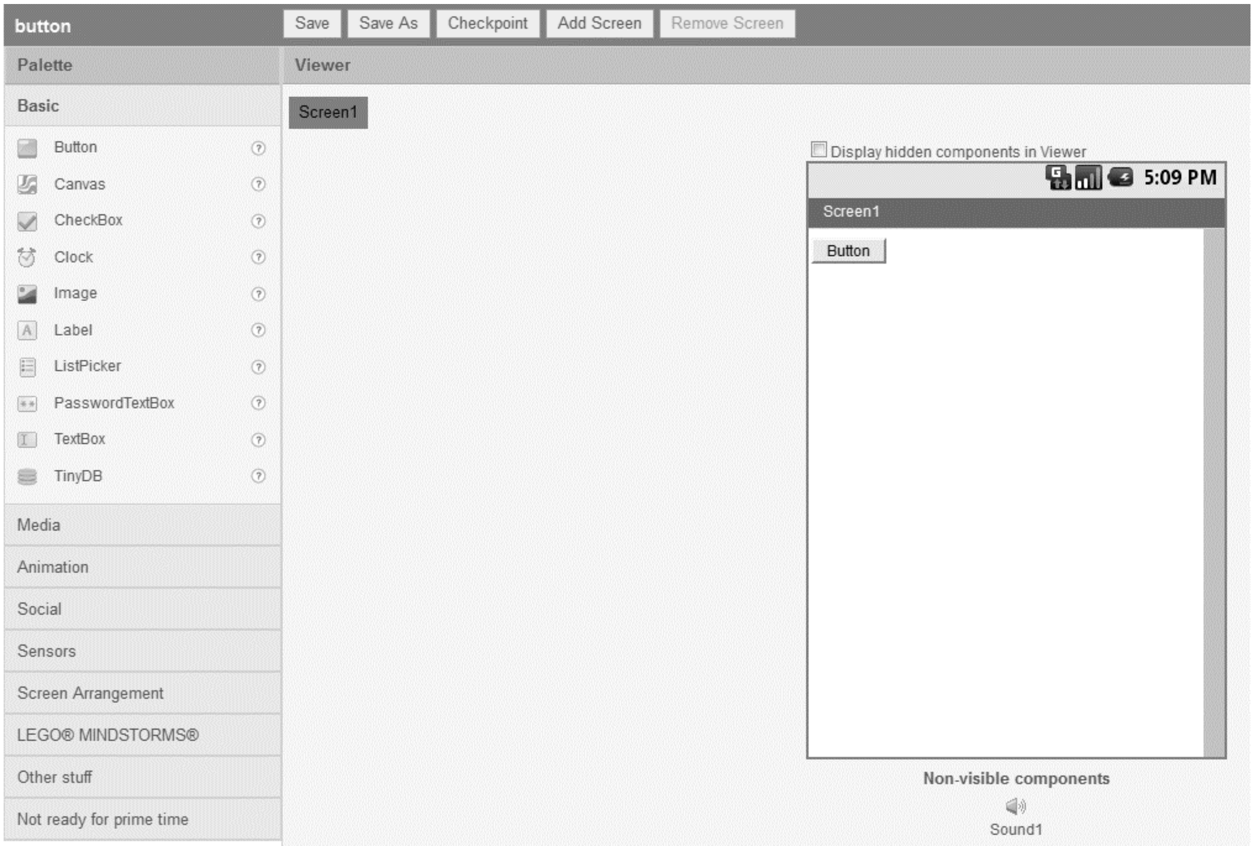
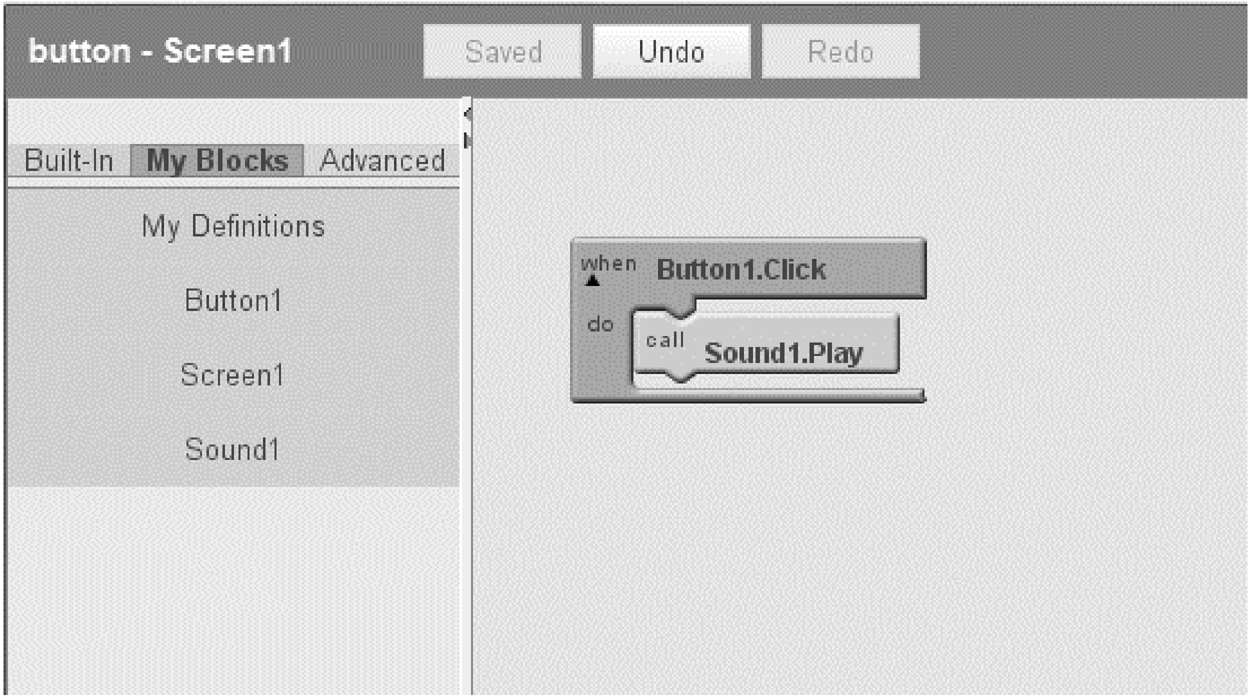


Figure 2. Screenshot of App Inventor Designer interface.



```
when Button1.Click
do  call Sound1.Play
```

Figure 3. Screenshot of App Inventor Blocks Editor interface.

Section of Conventional Worked Example

1. Click on "My Blocks" to see the blocks for components you created.
2. Click on "clap"
3. Drag out a *when clap.Touched* block
4. Click on "clapSound"
5. Drag out a *call clapSound.Play*
6. Connect it after the *when clap.Touched*
7. Click on "My Blocks" and then on "AccelerometerSensor1"
8. Drag out *when AccelerometerSensor1.AccelerationChanged*.

Section of Subgoal Labeled Worked Example

Handle Events from My Blocks

1. Click on "My Blocks" to see the blocks for components you created.
2. Click on "clap"
3. Drag out a *when clap.Touched* block

Set Output from My Blocks

4. Click on "clapSound"
5. Drag out a *call clapSound.Play*
6. Connect it after the *when clap.Touched*

Handle Events from My Blocks

7. Click on "My Blocks" and then on "AccelerometerSensor1"
8. Drag out *when AccelerometerSensor1.AccelerationChanged*.

Figure 4. Sample of worked examples used by participants in sessions.

Placement	Assessment Type	Transfer Type	Task
End of Session 1	Immediate	Procedural	Write the steps you would take to permanently change the color of the canvas black.
End of Session 1	Immediate	Component	Write the steps you would take to create a block that plays the clapSound when the top of the phone is tilted down (positive YAccel value).
End of Session 1	Immediate	Procedural	Write the steps you would take to make the "clap" ImageSprite move 5 pixels to the right of its current location when it is touched (hint: components called "ImageSprite.X" deal with the x coordinates of an ImageSprite).
End of Session 1	Immediate	Component	When playing long sound clips instead of short sounds clips, it's better to use the player component than the sound component. Write the steps you would take to add a melody to your Music Maker app and make it play when touched (create component and block).
End of Session 1 (not used in Exp 3)	Immediate	Procedural	Write the steps you would take to make the screen change colors depending on the orientation of the phone; specifically, the screen turns blue when the pitch is greater than 2 (hint: you'll need to make an orientation sensor and use blocks from "Screen 1" in My Blocks).
Start of Session 2	Delayed	Component	Write the steps you would take to add a tambourine to your Music Maker app.
Start of Session 2 (not used in Exp 3)	Delayed	Procedural	Write the steps you would take to make a PasswordTextBox that said "Type password here" when there was no text in it, and the password is appinventor.
Start of Session 2	Delayed	Component	Write the steps you could take to make the drum1 sound play twice when drum1 is touched.
Start of Session 2 (not used in Exp 3)	Delayed	Procedural	Write the steps you would take to make your tambourine play when the phone is shaken at an x value greater than 2.
End of Session 2	Immediate	Component	Write the steps you would take to italicize the fortune presented.
End of Session 2	Immediate	Procedural	You can create a ball that moves around your screen at a set heading (in degrees, 0 degrees is towards the right, 90 degrees is towards the top), set interval (in milliseconds), and set speed (in pixels). Write the steps you would take to make a ball that moves at a rate of 5 pixels every 250 milliseconds towards the right of the screen (hint: animation components must be on a canvas).
End of Session 2	Immediate	Component	Write the steps you would take to create a list of colors and make the ball to change to a random color whenever it collided with something.
End of Session 2	Immediate	Procedural	Write the steps you would take to make the ball change direction (called heading in App Inventor) to 90 degrees more than its current direction whenever it is touched.
End of Session 2 (not used in Exp 3)	Immediate	Component	Write the steps you would take to create an app using cowbell.jpg and cowbell.wav so that the sound plays when the picture was touched or when the phone is shaken at a y value greater than 2.

Figure 5. Problem solving tasks completed by participants in the order that participants received them.

	Component Transfer	Procedural Transfer
Instruction	From “My Blocks” click on "drumSound" and drag out *call drumsound.Play* and put it next to the "then-do" part of the *if* block.	From “My Blocks” click on "clapSound" and drag out a *call clapSound.Play* and connect it after the *when clap.Touched*
Assessment	From “My Blocks” click on "clapSound" and drag out *call clapsound.Play* and put it next to the "then-do" part of the *if* block.	From “My Blocks” click on "clap" and drag out a *set clap.X* and connect it after the *when clap.Touched*

Figure 6. Example of component and procedural transfer between instruction and assessment for “Set output in My Blocks” subgoal.

Assessment task: You can create a ball that moves around your screen at a set heading (in degrees, 0 degrees is towards the right, 90 degrees is towards the top), set interval (in milliseconds), and set speed (in pixels). Write the steps you would take to make a ball that moves at a rate of 5 pixels every 250 milliseconds towards the right of the screen (hint: animation components must be on a canvas).

Correct answer:

Drag out ball component → achieves function of creating component

$$\left.\begin{array}{l}\text{Set heading to 0}\\ \text{Set interval to 250}\\ \text{Set speed to 5}\end{array}\right\}\text{ achieves function of setting properties}$$

Participant answer, example 1:
Drag out canvas
Drag out ball
~~Change image of button to ball~~
Change speed to 5
Change interval to 250
Change heading to 0

Score for example 1: Participant received 2 points for the "correct" score because they achieved both functions. Participant also received 2 points for the "attempted" score because they attempted both functions.

Participant answer, example 2:
~~My blocks   screen1 set.screen1backgroundcolor to black~~
Drag a canvas to bottom of screen
Choose set canvas as,

```
        Call canvas1.drawcircle x t
                                  y number
                                  z number
```

Set t to change with a timer. Can't find the button

Score for example 2: Participant received 0 points for the "correct" score because they did not achieve either function. Participant received 2 points for the "attempted" score because they attempted to create a component to serve as the ball, and they attempted to set the properties of that component.

Figure 7. Example of participant responses to assessment question and scores for each response.

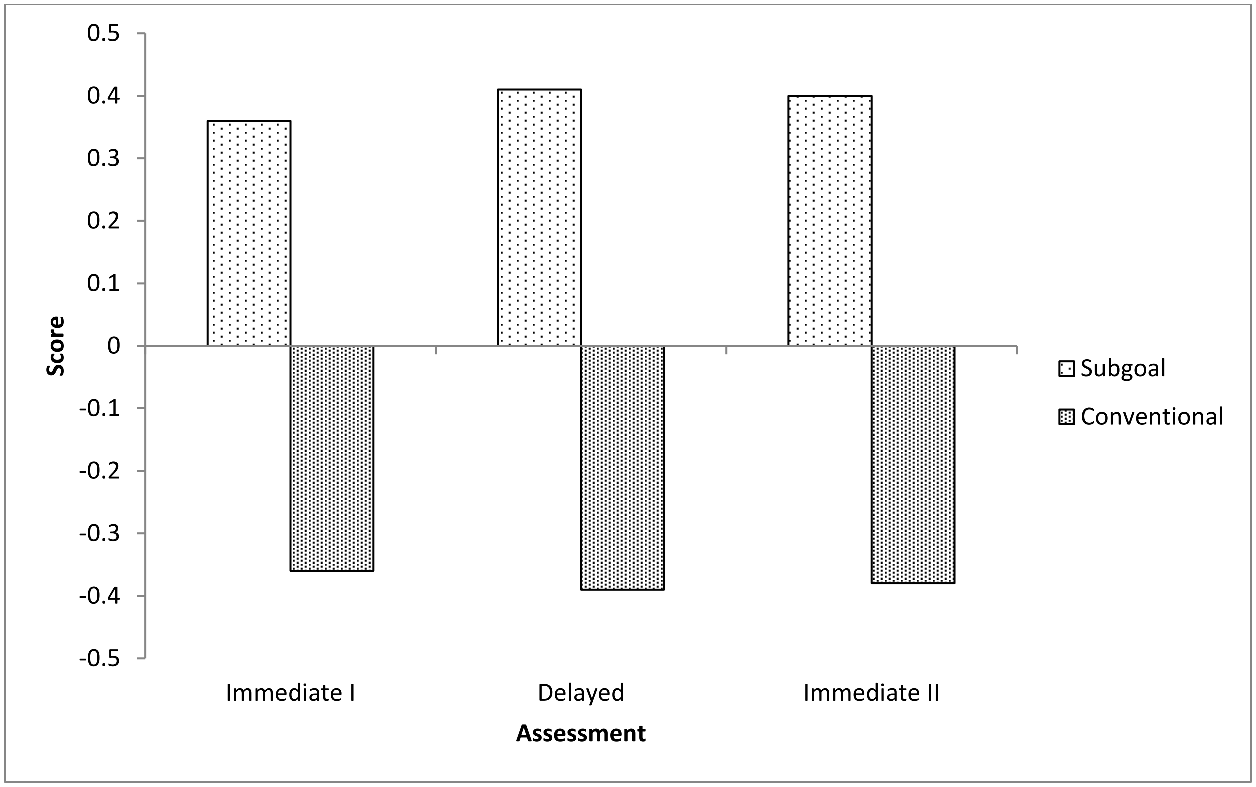


Figure 8. Scores for assessments in Experiment 1 for each group. The data were normalized to remove differences in scores due to differences in scale (i.e., maximum points for each assessment).