Subgoal Labeled Instructional Text and Worked Examples in STEM Education

A Thesis
Presented to
The Academic Faculty

by

Lauren Elizabeth Margulieux

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Psychology

Georgia Institute of Technology

May, 2014

Subgoal Labeled Instructional Text and Worked Examples in STEM Education

ACKNOWLEDGEMENTS

I wish to thank Gerin Williams and Alex Duncan, research assistants in the PSET laboratory, for help in developing materials, editing papers, and collecting and scoring data. I would also like to thank the American Psychological Foundation for the research grant that supported this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In science, technology, engineering, and mathematics (STEM) education, problem solving tends to be highly procedural, and these procedures are typically taught with general instructional text and specific worked examples. Instructional text broadly defines procedures for problem solving, and worked examples demonstrate how to apply procedures to problems. Subgoal labels have been used to help students understand the structure of worked examples, and this feature has increased problem solving performance. The present study explored using subgoal labels in instructional text to further improve learners' problem solving performance. A factorial design examined the efficacy of subgoal labeled instructional text and worked examples for programming education. The results of the present study suggest that subgoal labels in instructional text can help learners in a different way than subgoal labels in worked examples. Subgoal labels in text helped the learner articulate the general procedure better, and subgoal labels in the example helped the learner apply those procedures better. When solving novel problems, learners who received subgoal labels in both the text and example performed better than those who received subgoal labels in only the example. Learners who received subgoal labels in only the example performed better than those who received subgoal labels in only the text and those who did not receive subgoal labels at all. The present study indicates that subgoal labeled instructional text can improve novices' problem solving performance in programming, but subgoal labels must appear in both the text and example.

CHAPTER 1

INTRODUCTION

Knowledge of science, technology, engineering, and mathematics (STEM) subjects is increasingly necessary in our society. As STEM fields advance, individuals need to understand more about these fields generally to make well-informed decisions, such as those made when buying technology, and to understand technical information, such as that in a medical diagnosis (Committee on Highly Successful Schools or Programs in K-12 STEM Education (CHSSP), 2011). In addition, individuals with advanced STEM knowledge are needed to fill increasingly technical jobs and promote innovation (Katehi, Pearson, & Feder, 2009). Students, however, commonly have trouble in their STEM classes.

Research on STEM education in the United States (US) suggests that many American students will not reach the level of STEM literacy needed in society. Moreover, employers in many industries are filling an increasing portion of elite STEM positions in the US with international applicants because American applicants lack the necessary knowledge and problem solving skills needed to succeed (CHSSP, 2011). To address the deficit of qualified STEM specialists and general STEM literacy in society, interventions to improve STEM education are needed (e.g., Beatty, 2011).

1.1 Improving Transfer in STEM Education

A major learning goal for STEM students is that they understand core concepts and principles with the underlying expectation that they will be savvy consumers of information in that STEM discipline (Nielsen, 2011). For example, a person who has taken science classes should be able to generally assess the validity of scientific research and determine the utility of information provided by that research. If that person reads about a study that correlates heat with aggression, they should understand that heat does not necessarily cause aggression.

Worked examples are a popular tool to help students learn STEM procedures, but they can inhibit transfer because they are specific to a particular context (LeFevre & Dixon, 1986). Learners must be able to extract information from these specific examples that allows them to transfer their knowledge and solve problems in an unpredictable array of contexts (Committee on Developments in the Science of Learning (CDSL), 2000). Improvements in learning outcomes, such as problem solving performance, can be facilitated by redesigning instructional materials (Catrambone, 1990; Meyer & McConkie, 1973). The use of subgoal labels is an instructional design technique that has been effective for improving learner transfer.

To understand what a subgoal is, consider a complex problem solution. Achieving the solution would be the overall goal, and the problem solver takes many individual steps towards that goal. Subgoals are functional pieces of the overall solution achieved by completing one or more individual steps (for an example see Appendix C). The individual steps taken to achieve a subgoal can change based on problem solver or strategy, but the subgoals needed to complete a problem do not (Catrambone, 1994). Subgoals are specific to a class of problems within a domain but not to a single problem; therefore, a particular subgoal can be found in many problems within a topic. If learners are taught how to identify and achieve subgoals, their success at solving novel problems in that topic can increase (Catrambone & Holyoak, 1990).

Subgoal labels have been used to teach learners the subgoals of a class of problems. Subgoal labels are typically used in worked examples. Research on subgoal labeled worked examples suggests that improved outcomes caused by subgoal labels stem from three sources: highlighting the structure of the worked example for the learner, helping the learner mentally organize information, and inducing the learner to self-explain the examples (see Figure 1).

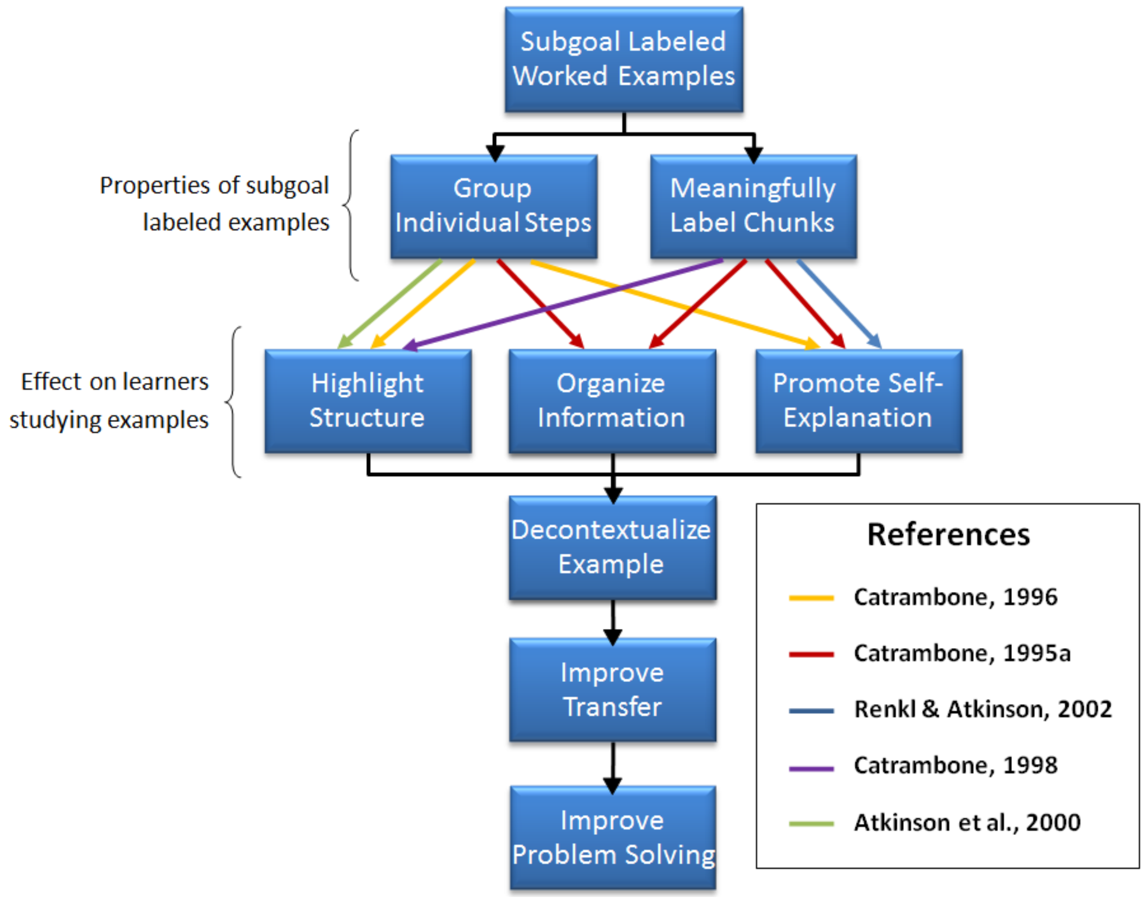


Figure 1. Diagram of how subgoal labeled worked examples can help learners improve problem solving performance. The level starting with "group individual steps" describes the physical characteristics of subgoal labels. The level starting with "highlight structure" describes how these characteristics help the learners use effective learning strategies.

When novices learn from worked examples, problem solving knowledge is usually encoded with superficial features of the problem that do not create the retrieval cues necessary for transfer (Bassok & Holyoak, 1989). Conversely, experts' knowledge of a domain is organized around high-level concepts and principles, allowing them to identify structural aspects of problems and transfer to new contexts more easily (CDSL, 2000). To help students learn problem solving procedures from worked examples, the structure of worked examples needs to be emphasized (Atkinson et al., 2000).

Subgoal labels group steps of an example by structural features, and learners who receive subgoal labeled examples are likely to chunk steps that are grouped (Catrambone, 1996). This chunking changes learners' mental representation of a problem solving procedure from individual steps to subgoals, reducing the demand on working memory (Atkinson & Derry, 2000; Catrambone, 1996). Moreover, this grouping emphasizes the example's structure, helping students transfer knowledge to solve other problems (Catrambone, 1998).

Even if novices recognize the structure of examples, they might need help to mentally organize new information (CDSL, 2000). For example, even though a student might recognize the subgoals in a math problem, they do not necessarily understand how those subgoals relate to each other or how they could be applied to a new problem. Helping learners create an organizational scheme in a domain is critical because how learners organize and interpret new information affects their proclivity to remember, use, and acquire new knowledge (CDSL, 2000). When subgoals in examples are meaningfully labeled, the student can see what function a group of steps achieves (Renkl & Atkinson, 2002). Catrambone (1995a) found that participants who received meaningful subgoal labels in worked examples tended to explain their solutions using those labels, suggesting that is how they mentally organized information. Better mental organization of information in a domain enables better transfer within the domain (CDSL, 2000).

A learning strategy that can be used to help students learn more effectively from worked example is self-explanation. Chi, Leeuw, Chiu, and LaVancher (1994) and Paas and van Gog (2009) argue that self-explanation helps learners to understand procedures rather than simply memorize them. The more learners understand a procedure, the better they can adapt and transfer it to novel problems (e.g., Catrambone, 1998). Moreover, Sweller (2010) argues that self-explanation guides cognitive resources to focus on relevant information. The amount of explanation a learner makes is typically self-driven (Chi et al., 1994), but self-explanation can also be prompted by the design of instructional materials (Catrambone, 1998).

By grouping steps of a worked example under a meaningful label, subgoal labels prompt the learner to self-explain how the steps are related to the label (Catrambone, 1995a). Furthermore, when the same subgoal label appears multiple times, students can compare methods for achieving that subgoal across several instances and develop a deeper understanding of the examples (Atkinson et al., 2000; Catrambone & Holyoak, 1990). Moreover, when subgoals are meaningfully labeled, they can reduce the number of incorrect self-explanations that students make (Renkl & Atkinson, 2002). Perhaps most important, though, is that the design of the worked examples can externally and consistently prompt students to make these self-explanations (Renkl & Atkinson, 2002).

1.2 Using Subgoal Labels in Instructional Text

Subgoal labels have been used primarily in worked examples (e.g., Catrambone, 1998; Renkl, 2002). The impact of subgoal labels in instructional text has not been explored. Instructional text is defined as general descriptions of how a procedure is done

(Biederman & Shiffrar, 1987, LeFevre & Dixon, 1986), and it "is intended to communicate a certain set of skills for reasoning or thinking cogently within that field," (Reder & Anderson, 1980; p. 121). Instructional text is important for learning procedures and can improve performance and transfer on procedural tasks. For example, Smith and Goodman (1984) found that participants who received structural or functional information about the system on which they completed tasks read the steps faster, recalled them more accurately, and transferred their knowledge to a novel system better than participants who did not receive that information.

Instructional text that emphasizes the concepts and principles in a domain can help novices learn how to organize new information. For example, learners solved problems more successfully when they received concept-oriented equations (i.e., written to show the purpose of the equation) compared to calculation-oriented equations (i.e., written to expedite calculation; Atkinson et al., 2003). In addition, when learners received general instructions and principles for a domain, they transferred to novel tasks more successfully than participants who received instructions without principles (Catrambone, 1995b). These findings suggest that giving students concepts and principles around which to organize information helps them to perform better.

Subgoal labeled instructional text might help novices understand new problem solving procedures by providing extra guidance. However, worked examples are important because they provide information about how to apply principles to problem solving (Catrambone, 1998; Trafton & Reiser, 1993). If learners receive only subgoal labeled instructional text, they might have trouble applying that information to problem solving without subgoal labeled worked example to guide them.

General instructional text and specific worked examples can complement each other. For example, algebra education typically involves being taught a procedure, such as how to use an equation, and then being provided worked examples for how to solve problems using that procedure (Bassok, 1990). Subgoal labels might help text and examples become more complementary. Instructional text and worked examples that use the same subgoal labels might help learners connect information sharing the same label and integrate information presented in each type of instructional material.

Subgoal labels in instructional text could also help learners finding information in the text to help them resolve specific problem solving impasses. VanLehn, Jones, and Chi (1992) found that when participants had trouble with a physics problem, many participants spent a long time searching the text, but only a small proportion found information relevant to the problem. Subgoal labels in text might help students who are struggling with part of a problem find relevant information more quickly.

A number of ways that instructional materials can be designed to improve learners' abilities to transfer knowledge in STEM domains have been discussed. The design recommendations can be summarized as follows:

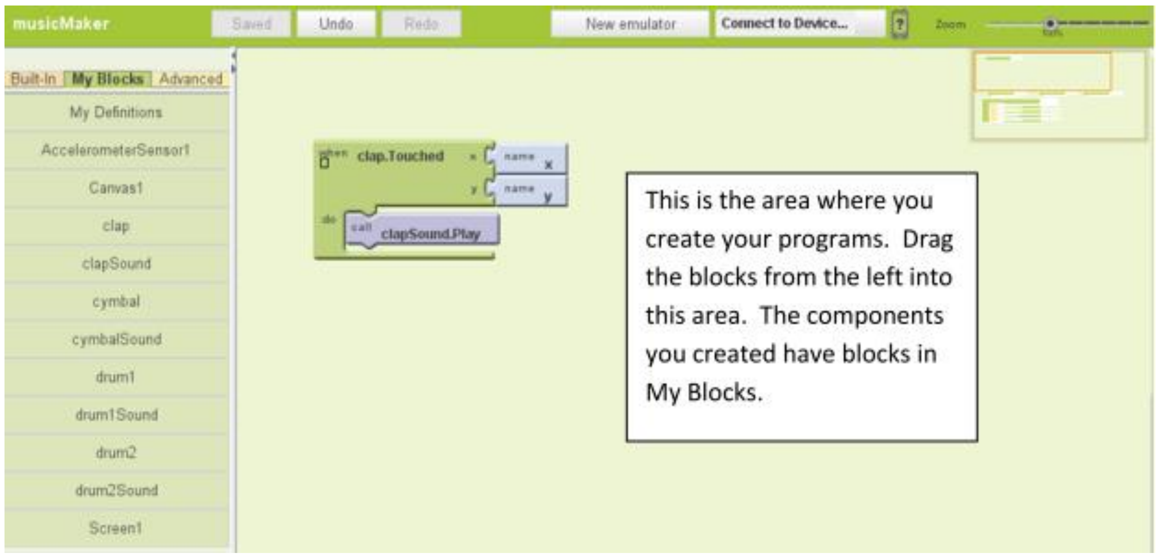
- Instructional materials should help learners recognize the structure of examples to help learners transfer knowledge and solve novel problems.
- Instructional materials should guide learners' mental organization of knowledge in new domains.
- Instructional materials should help learners understand instructions by encouraging learning strategies like self-explanation.

- Instructional materials should help learners integrate specific (e.g., worked examples) and general (e.g., instructional text) instructions.

The present experiment used subgoal labels to achieve these four recommendations simultaneously. For this reason, it was not possible to determine if any of the four recommendations are more effective than the others. The present experiment instead tested how effective subgoal labeled instructional materials are compared to unlabeled instructional materials. Further research would be required to disentangle the contribution of the four recommendations.

1.3 Present Study

The present study explored the effectiveness of subgoal labeled instructional materials compared to unlabeled instructional materials to teach computer programming. In the study, participants learned how to use Android App Inventor, a programming language for creating applications (apps) for an Android device, to create a Fortune Teller app. This computer programming language was chosen because it is a drag-and-drop language. Drag-and-drop programming languages are good tools for teaching novices because, instead of writing code to create programs, users drag components from a menu and piece them together like puzzle pieces (see Figure 2). This type of code creation is more easily understood by novices (Hundhausen, Farley, & Brown, 2009). Instructions from the ICE Distance Education Portal (Ericson, 2012) were used to develop instructional materials for the study. Instructional materials in all conditions were the same except for the subgoal labels. Subgoals were determined using the Task Analysis by Problem Solving (TAPS) procedure (Catrambone et al., 2013) and consultation with subject-matter experts.



This is the area where you create your programs. Drag the blocks from the left into this area. The components you created have blocks in My Blocks.

Figure 2. App Inventor interface with interlocking blocks of code used to program features.

Instructional materials included general text about how to create apps (i.e., instructional text) and a video demonstration and step-by-step guide showing how to create a Fortune Teller app (i.e., worked example). A video demonstration was used because it can be a quick and natural way for users to learn direct-manipulation interfaces (Palmiter, Elkerton, & Baggett, 1991) like Android App Inventor. Subgoal labels were presented in the video as callouts (see Figure 3). The step-by-step guide gave participants textual instructions for creating the Fortune Teller app. Participants completed a series of assessment tasks designed to measure their problem solving performance and their mental organization and representations of information learned.

It was predicted that participants who received the subgoal labeled worked example would solve problems better than those who did not. It was also predicted that participants who received the subgoal labeled instructional text would not necessarily solve problems better than those who did not. Instead, an interaction was predicted such that participants who received subgoal labels in both text and example would perform better than all other groups, and participants who received subgoal labels in only the text would perform equally to those who received no subgoal labels in the example or instructional text.

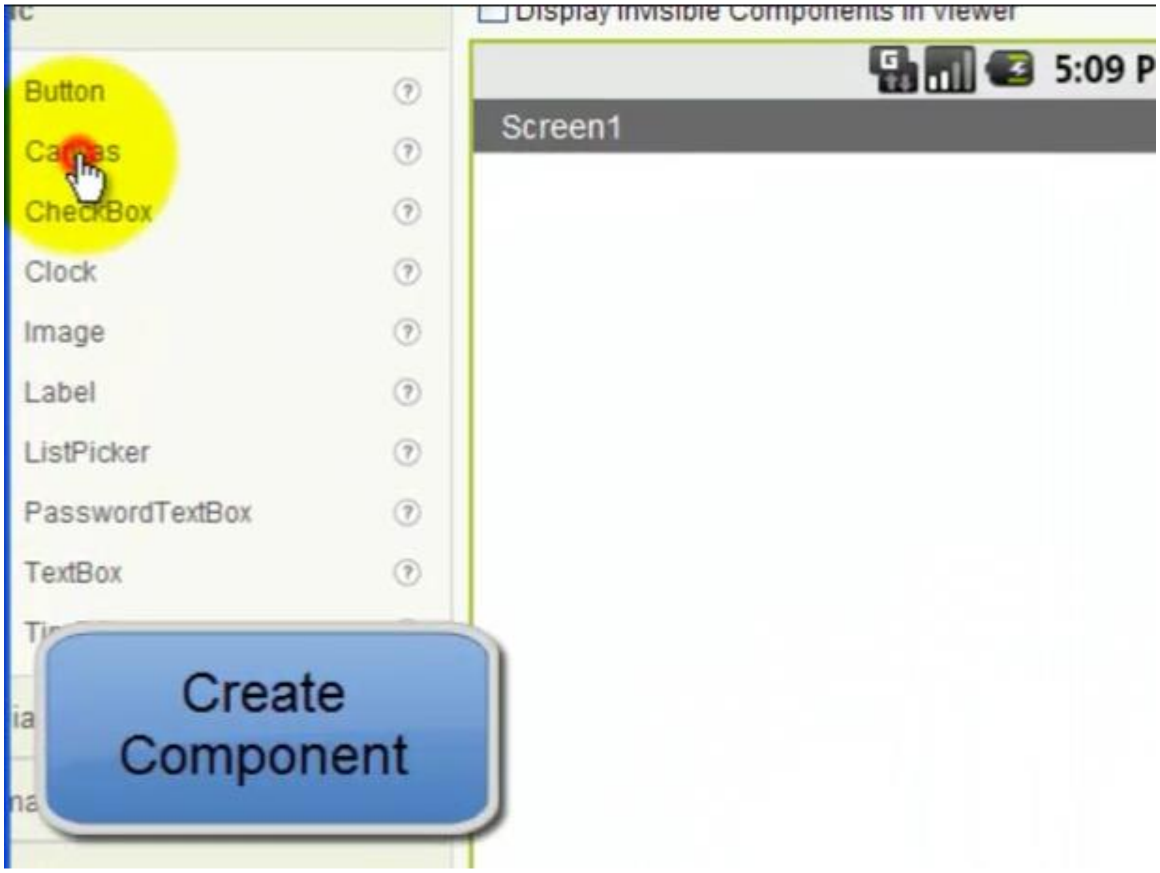


Figure 3. Screenshot of a subgoal callout in video demonstration.

CHAPTER 2

METHODOLOGY

2.1 Participants

Participants were 120 students from the Georgia Institute of Technology recruited through Experimetrix and advertisements in psychology classes. People were disqualified for participation if they had experience with Android App Inventor or had taken more than one high-school or college-level course in computer science or computer programming. These restrictions were necessary because instructions were designed for novices. In previous experiments using similar instructional materials, there was not a statistical difference in scores between participants who had taken one course in computer science and those who had not taken any courses in computer science (Margulieux, Guzdial, & Catrambone, 2012).

2.2 Procedure

Sessions lasted between 70 and 90 minutes depending on how quickly participants completed each section. First, participants filled out a demographic questionnaire to provide information about their age, gender, academic field of study, SAT scores, high school and college GPA, year in school, number of completed credits, primary language, computer science experience, comfort with computers, and expected difficulty of learning App Inventor. These factors were collected because they are possible predictors of performance in computer science (Rountree, Rountree, Robins, & Hannah, 2004).

Next was the instructional period. During the instructional period, participants received the instructional text about how to create an app generally and the worked example (i.e., a video demonstration and step-by-step guide for the Fortune Teller app). Instructional materials can be found in Appendices A, B, C, and D. Participants had up to 30 minutes to create the app using the Android App Inventor website. 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?”).

Following the instructional period was the assessment period. During this time, participants did not have access to the instructional materials, but they did have access to the App Inventor website and the app that they created. The participants were allowed access to the app that they created to serve as a memory cue to aid problem solving.

The assessment tasks included 1) a problem solving task, 2) an explanation task, 3) a card sorting task, and 4) a generalization task. The problem solving task asked participants to list the steps that they would take to modify or add features to their Fortune Teller app (see Appendix E). This assessment was meant to measure how well participants could solve novel problems with App Inventor. Participants were limited to a maximum of 25 minutes on this task, so, similar to an exam, they were not permitted to work on the problems for an indefinite amount of time. Next, in the explanation task, participants were given the correct solutions from the previous problem solving task and asked to group steps of the problem solutions (see Appendix F). Participants were then asked to label their groups to explain what each achieved. This assessment was meant to measure how well participants could group steps based on structural similarity and self-explain problem solutions.

The next assessment was a card sorting task. Participants were asked to sort the cards by similarity into a minimum of two and a maximum of six categories. This category limit was meant to encourage participants to create general categories that made abstract connections among cards and not to create specific categories that are technically correct but do not represent abstract concepts. For example, if participants were allowed to make as many categories as they wanted, then they could sort the cards by subgoals and apps instead of being forced to pick between the two schemes. An ideal scheme that categorized by subgoals regardless of app would suggest a structural understanding of the information, whereas a scheme that categorized by app regardless of subgoal would suggest a more superficial understanding.

To create the cards, the experimenter grouped the low-level steps that created three apps into subgoals, and then the steps that achieved one subgoal were put onto a card without a subgoal label (see Appendix H). Because subgoals are repeated within each app, there were several instances of each subgoal in each app (each instance of a subgoal was achieved by different low-level steps). Two instances of each subgoal from the apps were used (the entire app was not represented on the cards); there were a total of 30 cards – 10 from each of the 3 apps or 6 from each of the 5 subgoals (see Appendices G and H). The card sorting was meant to measure how well participants recognized structural commonalities among examples.

The generalization assessment asked participants to describe the general procedure that they would take to create an app (see Appendix I). An ideal response to this task would include the subgoals needed to make the app and exclude unnecessary details. This assessment was meant to measure how well participants could abstractly outline the problem solving procedure that they learned in the session.

2.3 Design

The experiment was a two-by-two, between-subjects, factorial design: the format of instructional text (subgoal labeled or unlabeled) by the format of the worked example (subgoal labeled or unlabeled). In each of the conditions, participants received the same content in the instructional text and the worked example, but the presence of subgoal labels differed. The dependent variables were performance on the tasks (to determine participants' knowledge organization and effectiveness in solving problems), minutes spent completing the tasks (to determine participants' efficiency), and minutes spent looking at each type of instruction.

CHAPTER 3

RESULTS AND DISCUSSION

Of the demographic information collected as possible predictors, two were correlated with performance. There was a negative correlation between SAT Writing scores and time spent on the instructional period, *r* = -.28, *p* = .022. Participants' with higher SAT Writing scores tended to finish the instructional period faster. There was also a positive correlation between participants' subjective ratings of their comfort with computers and number of attempted problem solutions, *r* = .25, *p* = .009. There was not, however, a statistically significant correlation between comfort with computers and number of correct problem solutions, *r* = .16, *p* = .341. These predictors are not expected to confound the analyses of the performance metrics because there are no differences among experimental conditions on these predictors (see Table 1). This finding indicates that the variance is evenly distributed, and, therefore, no group would have an advantage.

Table 1: *Distribution of Selected Demographics Among Conditions*

	SAT Writing				Comfort with Computers			
Condition	*M*	*SD*	*r*	*p*	*M*	*SD*	*r*	*p*
Subgoal-text, Subgoal-example	631	87	-.07	.73	5.88	1.16	-.11	.68
Unlabeled-text, Subgoal-example	659	54	.03	.84	6.41	.93	.08	.78
Subgoal-text, Unlabeled-example	625	72	-.31	.08	5.57	1.19	.02	.95
Unlabeled-text, Unlabeled-example	665	41	-.20	.26	6.08	1.05	.41	.15

Note: Comfort with computers on 7-pt. scale (1-Not Comfortable At All and 7-Very Comfortable).

3.1 Problem Solving Performance

To score the problem solving assessment, participants' solutions were compared to the correct solutions for each problem. Participants earned one point for each correct step they took towards the problem solution. This scoring scheme afforded more sensitivity than judging an entire solution as correct or incorrect. The maximum score that participants could earn was 22 for completing all four problem solving tasks correctly. Participant responses were scored by two raters, and interrater reliability was measured with an intraclass correlation coefficient of absolute agreement (ICC(A)). ICC(A) for this assessment was .94.

As hypothesized, there was a main effect of example design consistent with previous literature (e.g., Margulieux et al., 2012): participants who received subgoal labels in the example (*M* = 13.1, *SD* = 6.0) performed better than those who did not (*M* = 5.5, *SD* = 4.8), *F* (1, 116) = 70.19, *MSE* = 24.47, *p* < .001, est. ω^2 = .32, *f* = .76. A main effect of text design was also found: participants who received subgoal labels in the text (*M* = 11.0, *SD* = 7.1) performed better than those who did not (*M* = 7.6, *SD* = 5.7), *F* (1, 116) = 13.90, *MSE* = 24.47, *p* < .001, est. ω^2 = .06, *f* = .34. In addition, there was an ordinal interaction between text design and example design for the problem solving assessment, *F* (1, 116) = 12.82, *MSE* = 24.47, *p* = .001, est. ω^2 = .05, *f* = .57. This interaction shows that participants who received subgoal labels in the text performed better than those who did not but only when they also received subgoal labels in the example. This pattern suggests that the interaction caused a main effect of text, and closer evaluation showed that there was no simple main effect (see Figure 4 and Table 2).

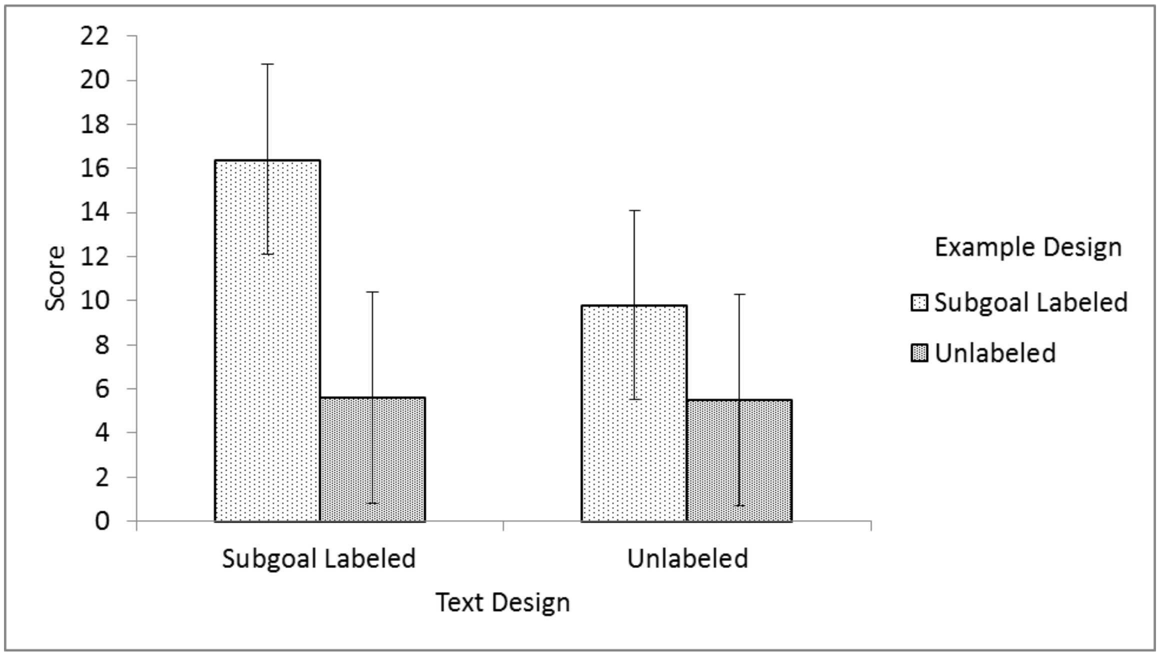


Figure 4. Scores on problem solving task by condition.

Table 2: *T-tests Comparing Groups for Problem Solving Task Score*

Condition	*n*	*M*	*SD*	*t*	Std. error	*P*
Subgoal-text, Subgoal-example	30	16.4	4.3			
				5.08	1.30	<.001
Unlabeled-text, Subgoal-example	30	9.8	5.6			
				3.18	1.36	.002
Subgoal-text, Unlabeled-example	30	5.6	4.8			
				.106	.133	.916
Unlabeled-text, Unlabeled-example	30	5.5	4.9			

Several studies (e.g., Atkinson et al. 2003; Catrambone, 1998; Renkl, 2002), including a study using similar instructional materials (Margulieux et al., 2012), have demonstrated that subgoal labeled examples help participants learn procedures in a way that allows them to transfer their knowledge to solve novel problems. The primary explanation for this effect is that subgoal labeled examples might help participants to learn the subgoals necessary to solve problems in a domain, which improves their problem solving performance in that domain (Catrambone, 1998). Other explanations for this effect suggest that subgoal labels promote self-explanation of worked examples (Renkl & Atkinson, 2002) and help learners transfer knowledge to solve novel problems (e.g., Catrambone & Holyoak, 1990).

Despite the benefits of subgoal labels in worked examples, a simple main effect of subgoal labels in instructional text was not expected or found. Learners in procedural domains rely on worked examples to show how to apply domain knowledge to problem solving (e.g., LeFevre & Dixon, 1986), so subgoal labeled instructional text might not have provided enough information to help students apply subgoals to problem solving. However, the interaction between text design and example design demonstrates that subgoal labeled text can improve problem solving performance when paired with subgoal labeled examples.

There are at least two reasons why participants receiving the subgoal labels in both text and examples performed better than participants in the other conditions. Having subgoal labels in both types of instructional material could have helped participants integrate the general information in the text with the specific information in the examples, leading to better understanding of the subgoals (VanLehn et al., 1992). Additionally, receiving the subgoal labeled text, similar to receiving principles in text (Bassok &

Holyoak, 1989), might have helped participants organize information from the general procedure better. Better organization of the general procedure could lead to more effective processing of the example if the same labels are used.

3.1.1 Attempted Problem Solutions

To better understand participants' performance, the problem solving task was also scored in terms of how much of the solution participants attempted. This score is meant to measure how many functional components of the solution the participants attempted, regardless of whether their answers were correct. Attempting components of the solution would suggest that a participant recognized the components needed in the solution, even if they could not achieve that component. To create an attempted score, the correct solutions for the problem solving tasks were deconstructed into the subgoals, or functional components, that were necessary to complete the solution. Participant solutions earned a point for each subgoal that was attempted. Attempting a subgoal was operationally defined as listing at least one step required to complete the subgoal, listing a step that would achieve a similar function (e.g., for a "set properties" subgoal, listing a step to change a property regardless of whether it was the correct property), or describing the subgoal. The maximum score that participants could earn was 10. ICC(A) for this assessment was .95.

There was a main effect of example design for the attempted score. Participants who received the subgoal labeled example (M = 6.9, SD = 2.7) attempted more subgoals than those who did not (M = 4.1, SD = 2.8), F (1, 116) = 30.43, MSE = 7.73, p < .001, est. ω^2 = .20, f = .50. No other statistically significant differences were observed (see Table 3). This result suggests that participants who received the subgoal labeled example recognized the necessary components of the task solutions better than those who did not, regardless of whether they could solve the problem correctly. Additionally, there was no effect of subgoal labels in text on the number of attempted subgoals which might be due to difficulty applying general procedures to specific problems (e.g., LeFevre & Dixon, 1986). These results in conjunction with problem solving performance suggest that the subgoal labeled text did not prompt participants to attempt more components but, when paired with the subgoal labeled example, helped them to get more of their steps correct for the components attempted.

Table 3: *T-tests Comparing Groups for Problem Solving Attempted Score*

Condition	*n*	*M*	*SD*	*t*	Std. error	*P*
Subgoal-text, Subgoal-example	30	7.0	2.6			
				.527	.696	.600
Unlabeled-text, Subgoal-example	30	6.7	2.8			
				3.42	.711	.001
Subgoal-text, Unlabeled-example	30	4.2	2.8			
				.496	.739	.622
Unlabeled-text, Unlabeled-example	30	3.9	3.0			

3.1.2 Time on Task

The amount of the time that participants spent working on the problem solving task was also measured. The majority of participants (75%) used the entire 25 minutes, but despite this range restriction, there were main effects of text and example design for time on task. Participants who received the subgoal labeled example (*M* = 20.5, *SD* = 3.0) completed the task faster than those who did not (*M* = 21.7, *SD* = 2.8), *F* (1, 116) = 7.88, *MSE* = 7.84, *p* = .006, est. ω^2 = .06, *f* = .26. Additionally, participants who received the subgoal labeled text (*M* = 20.25, *SD* = 2.2) completed the task faster than those who did not (*M* = 21.8, *SD* = 3.4), *F* (1, 116) = 9.19, *MSE* = 7.84, *p* = .003, est. ω^2 = .07, *f* = .28

(see Figure 5). There was no interaction of text and example design, *F* (1, 116) = .15, *MSE* = 7.84, *p* = .70. These findings suggest that receiving more instructional materials with subgoal labels resulted in less time on the task. Moreover, when paired with the performance results, these findings show that participants who performed better also completed the task faster. These results defy the typical tradeoff between speed and accuracy and suggest that participants who received subgoal labels in both text and examples were better problem solvers than those who did not.

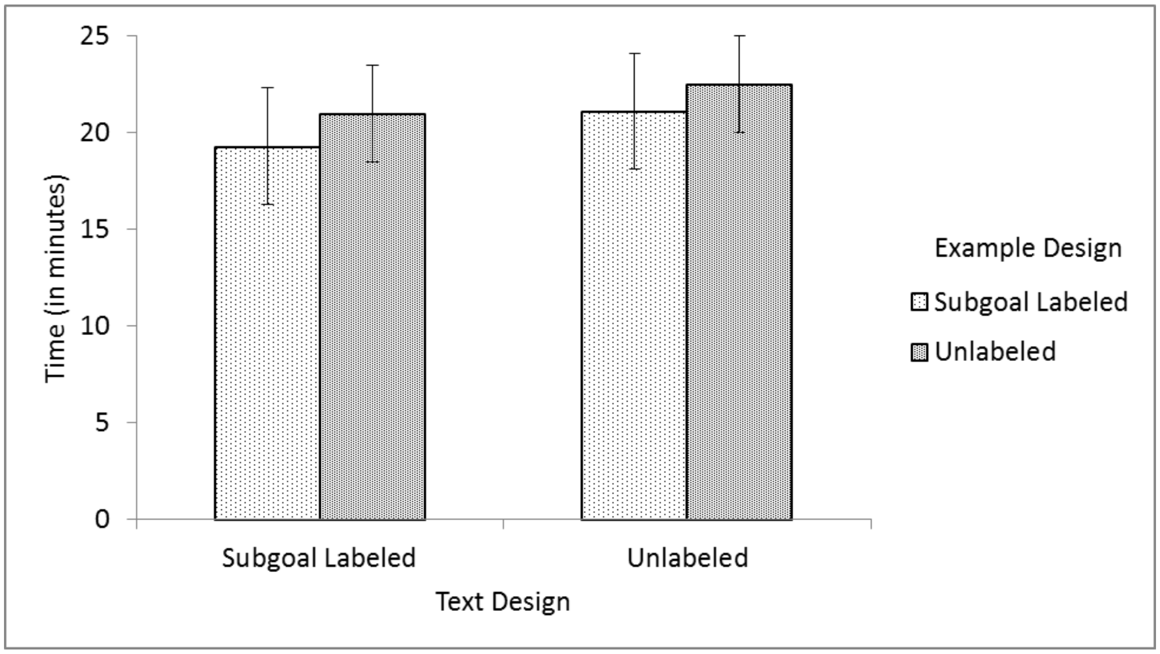


Figure 5. Time spent on problem solving task by condition.

The findings from the problem solving task provide two important pieces of information about subgoal labeled instructional materials. First, they demonstrate that subgoal labeled text needs to be paired with subgoal labeled examples to improve performance. Second, they show that subgoal labels can lead to better problem solving when the labels appear in both example and text than when subgoal labels appear only in examples.

It is possible that if a learner receives more subgoal labels, then increased exposure to the labels, independent of the type of instructional material that use them, could result in better problem solving performance. If this is the case, then receiving subgoal labels in text or additional subgoal labeled examples would produce the same results. Though additional research would be needed to test this possibility directly, the other measures in the present study suggest that subgoal labels have a different effect on learners when presented in instructional text than in worked examples.

3.2 Other Measures

3.2.1 Time on Instruction

The amount of time that participants spent using instructional materials in the instructional period was recorded. There was a main effect of example design: participants who received the subgoal labeled example (M = 20.9, SD = 3.26) finished the instructional period faster than those who did not (M = 23.7, SD = 4.69), F (1, 116) = 12.62, MSE = 16.83, $p < .001$, est. ω2 = .10, f = .32. There was no main effect for text design, F (1, 116) = .25, MSE = 16.83, p = .62, and there was no interaction of text and example design, F (1, 116) = .69, MSE = 16.83, p = .69. This effect could be the result of the subgoal labels helping participants to chunk the steps of the step-by-step together. Chunking steps could have help participants to remember more steps to complete in the App Inventor interface before referring back to the guide, and the labels could also have helped participants find their spot in the guide faster when they did refer to it.

3.2.2 Explanation Task

The participants completed an explanation assessment to measure how well they could organize and explain problem solutions. To do well on this assessment, participants did not need to solve problems, but they did need to recognize the steps of the solution that were structurally similar and explain why they were similar. Participants received two scores for this assessment: a grouping score for how well they organized steps and a labeling score for how well they explained groups. To score the grouping portion of this task, participants received points for grouping together structurally similar steps. For each group that contained only structurally similar steps, participants received one point, and they could earn up to nine points. ICC(A) for this assessment was .97.

Participants who received subgoal labels in both the text and example made more correct groups than all other conditions, and there were no other statistically significant differences (see Table 4). These results suggest that people who received subgoal labels in both the text and example were better at grouping the problem solutions into structurally similar chunks than the rest of the participants. To perform well on this task, participants need to integrate general procedural knowledge (to identify high-level groups) and specific application knowledge (to apply the groups to specific problems), and subgoal labels in both types of instructional material might have aided this integration.

Table 4: *T-tests Comparing Groups for Number of Groups Containing Structurally Similar Steps in Explanation Task*

Condition	*n*	*M*	*SD*	*t*	Std. error	*P*
Subgoal-text, Subgoal-example	30	4.8	2.5			
				2.51	.571	.015
Unlabeled-text, Subgoal-example	30	3.3	1.9			
				.060	.552	.952
Subgoal-text, Unlabeled-example	30	3.3	2.3			
				.122	.546	.903
Unlabeled-text, Unlabeled-example	30	3.2	1.9			

To score the labeling portion of this task, the labels that participants used to describe the groups were analyzed qualitatively to determine if participants correctly identified the purpose of the chunks. Over 50% of the responses given by participants who received subgoal labeled text correctly described the function of a group of steps, demonstrating a structural understanding of the solution. In contrast, less than 10% of the responses given by participants who received unlabeled text correctly described the function of steps. There was no meaningful difference for example design; participants in the subgoal labeled and unlabeled example groups each produced about 30% functional labels. The content of incorrect responses included superficial information such as how the blocks of code were pieced together or where in the interface the steps were completed. These results suggest that the participants who received subgoal labeled text were better than those who did not at explaining the solutions in a functional way.

For time on task, the only statistical difference between conditions was within the group that did not receive subgoal labeled text. People who received subgoals only in the worked example (*M* = 4.7, *SD* = 2.1) completed the explanation task faster than people who did not receive subgoals in any of the instructional materials (*M* = 7.0, *SD* = 3.1), *t* (58) = -3.06, *p* = .004 (see Figure 6 and Table 5 for full pattern of results). These results indicate that people who performed best on this task (i.e., those who received subgoal labels in both the text and example) did not take longer to group and label the solutions with more accuracy than it took the other participants to group and label the solutions with less accuracy.

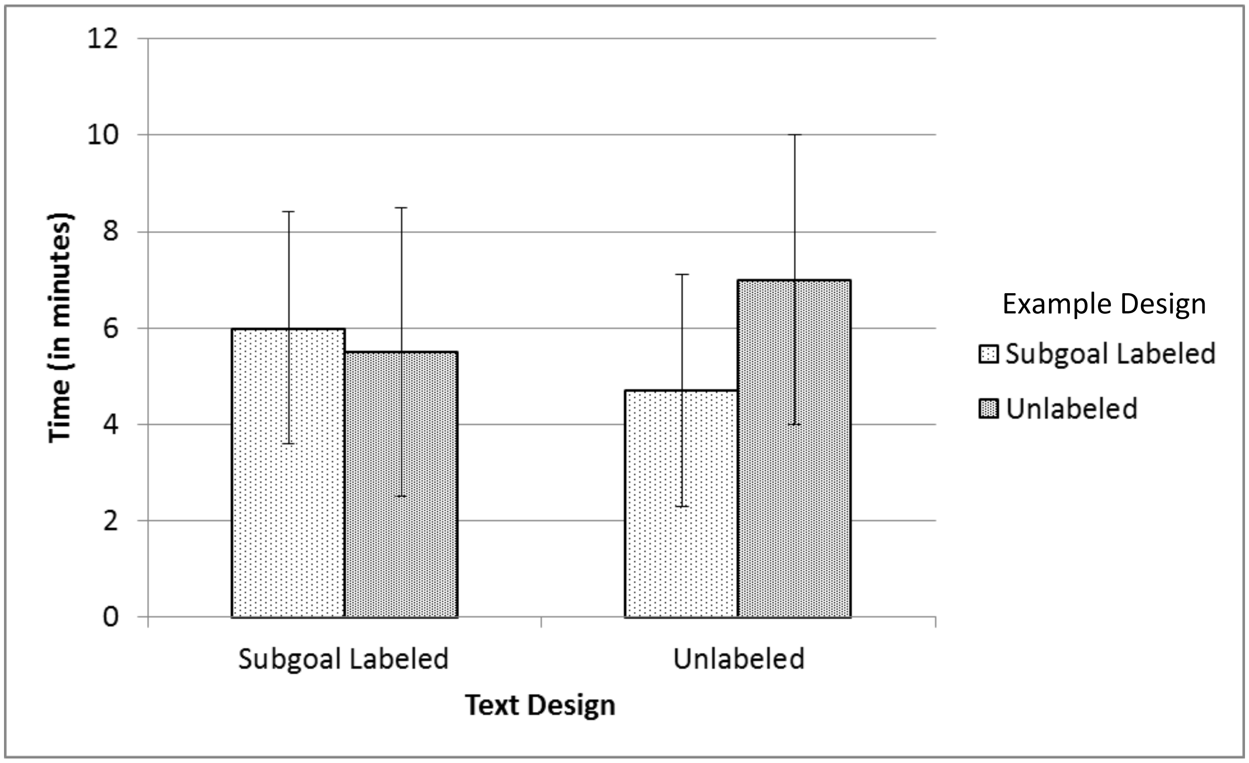


Figure 6. Time spent on explanation task by condition.

Table 5: *T-tests Comparing Groups for Time on Task for the Explanation Task*

Condition	*n*	*M*	*SD*	*t*	Std. error	*P*
Unlabeled-text, Unlabeled-example	30	7.0	3.1			
				1.171	.820	.247
Subgoal-text, Subgoal-example	30	6.0	2.6			
				.707	.724	.482
Subgoal-text, Unlabeled-example	30	5.5	2.7			
				1.194	.643	.237
Unlabeled-text, Subgoal-example	30	4.7	2.1			

This pattern of results suggests that the subgoal labeled instructional text has a different effect on learners than the subgoal labeled worked examples. The results of the labeling task suggest that subgoal labels in the text lead to better articulation of the purpose of steps. Better articulation can be a result of better self-explanation (Chi, 2009; Hill & Levenhagen, 1995) suggesting that the subgoal labeled text might prompt learners to self-explain the text.

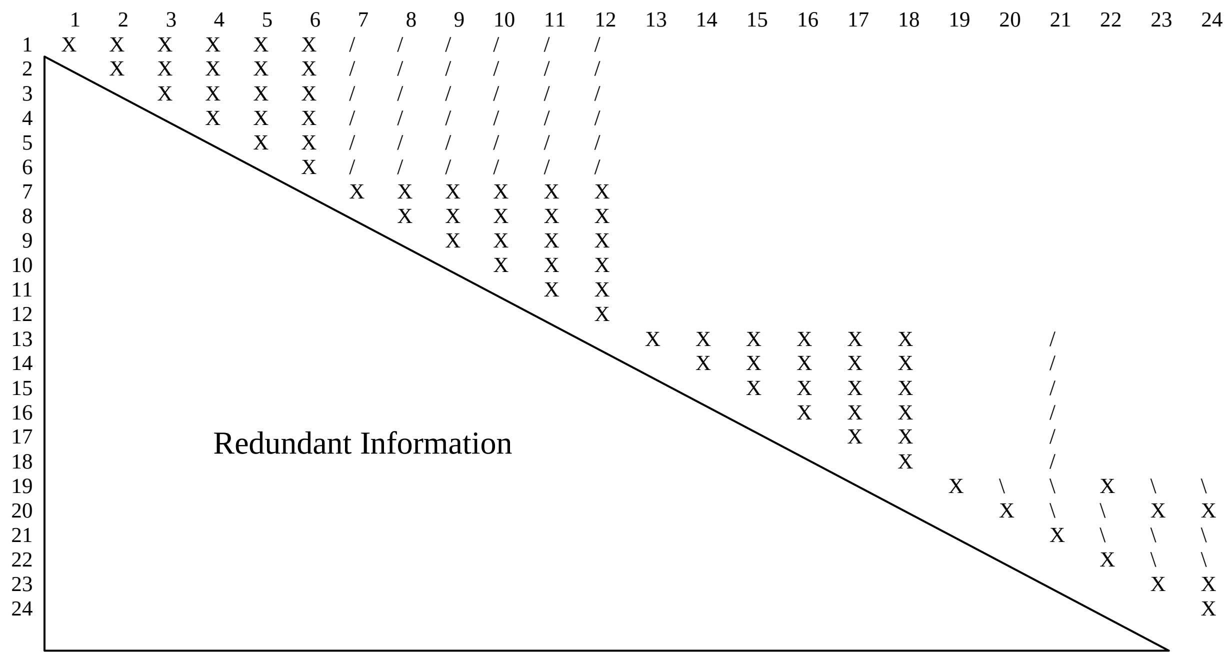
Though subgoal labeled text appears to promote self-explanation similar to subgoal labeled examples, the subgoal labeled text led to better articulation, and the subgoal labeled example did not. This finding suggests that the self-explanation induced by the instructional text might be different from the self-explanation induced by examples. Self-explanation of text might lead to better articulation of a general procedure. On the other hand, self-explanation of an example might lead to decontextualizing that example and allowing for transfer without the learner necessarily being able to describe the procedure.

3.2.3 Card Sorting Task

To further measure how well participants could classify parts of problem solutions, participants completed a card sorting task. This task is based on Chi, Feltovich, and Glaser's (1981) research that showed physics experts grouped problems based on procedural features (e.g., equation needed to solve problem) and novices grouped problems based on surface features (e.g., problems that included ramps). Though none of the participants in the present study were experts, for someone who has organized knowledge based on procedural features, these features would be weighted more than surface features (i.e., features of the app) leading to a more structural categorization scheme. To score this task, participant categorizations were compared to an ideal categorization. In the ideal categorization, cards representing the same subgoal would be grouped regardless of which app it represents. This categorization was considered ideal because it would best indicate that the participant understood the structural similarities in components across apps.

Agreement matrices based on Ferguson, Kazi, and Durso (2012) were used to compare the participants' categorizations to the ideal categorization. A matrix was made for each participant. To create the agreement matrix, each card appeared in the matrix, once as a row and once as a column. For participants' categorization, a forward slash, "/", denoted when a row item was placed in the same category as a column item. For example, if a participant placed cards 1 and 2 in the same category, then when card 1's row intersected card 2's column, a "/" would indicate that the cards were in the same category. For the ideal categorization, a back slash, "\", denoted the intersection. Thus, when the participants' and the ideal categorizations matched, the slashes would make an "X" to symbolize agreement (see Figure 7). Disagreement was indicated when cells had only a "/" or "\".

The matrices were analyzed using nonmetric multidimensional scaling. Multidimensional scaling maps items (e.g., cards) in *p* dimensions in such a way that less distance between items corresponds to more similarity between items. PROSCAL was used to create a similarity matrix for the ideal categorization and an average similarity matrix for each experimental condition by using an ordinal transformation of the agreement matrices. PROSCAL was also used to determine the number of dimensions in the ideal categorization and each condition's average categorization. For all



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	X	X	X	X	X	X	/	/	/	/	/	/												
2		X	X	X	X	X	/	/	/	/	/	/												
3			X	X	X	X	/	/	/	/	/	/												
4				X	X	X	/	/	/	/	/	/												
5					X	X	/	/	/	/	/	/												
6						X	/	/	/	/	/	/												
7							X	X	X	X	X	X												
8								X	X	X	X	X												
9									X	X	X	X												
10										X	X	X												
11											X	X												
12												X												
13													X	X	X	X	X	X			/			
14														X	X	X	X	X			/			
15															X	X	X	X			/			
16																X	X	X			/			
17																	X	X			/			
18																		X			/			
19																			X	\	\	X	\	\
20																				X	\	\	X	X
21																					X	\	\	\
22																						X	\	\
23																							X	X
24																								X

Figure 7. Agreement matrix for one participant. "X"s and blank cells indicate agreement between participants' and the ideal categorizations. "/"s and "\"s indicate disagreement. In this example, the participant grouped cards 1-12; 13-18 and 21; 19 and 22; 20, 23, 24; and 25-30 (not shown). The ideal categorization grouped cards 1-6, 7-12, 13-18, 19-24, and 25-30.

categorizations three dimensions was optimal because it passed the tests for determining dimensionality. The stress (i.e., measure of fit) was less than .1 (see Table 6); the elbow (i.e., the point that differentiates meaningful changes from insignificant changes in the variance accounted for) in Cattell's scree plot test was at three; adding a fourth dimension did not account for significantly more error.

Because the orientation of each matrix was arbitrary, orthogonal Procrustes rotation was used to match the orientation of condition matrices to the ideal matrix. Each of the conditions' matrices was then compared to the ideal matrix to determine the square root of mean squared differences per dimension. Though no null hypothesis significance test for the square root of mean square differences currently exists, this value describes how different two matrices are. None of the condition matrices were similar (i.e., had a low square root of mean square differences) to the ideal matrix (i.e., the matrix representing a structural organization) on any of the three dimensions. Furthermore, no meaningful differences among the condition matrices were found (see Table 6). For time on task, there were also no statistical differences between groups for this assessment (see Table 7).

Table 6: *Stress for Similarity Matrices and Square Root of Mean Squared Differences between Ideal Matrix and Condition Matrices per Dimension*

Matrix	Stress	Dimension 1	Dimension 2	Dimension 3
Ideal	.073	X	X	X
Subgoal-text, Subgoal-example	.068	.51	.46	.45
Unlabeled-text, Subgoal-example	.075	.48	.46	.47
Subgoal-text, Unlabeled-example	.074	.44	.45	.43
Unlabeled-text, Unlabeled-example	.070	.48	.44	.50

Table 7: *F-tests Comparing Time Spent on the Card Sorting Task*

Effect	*MS*	*F*	*p*
Text	16.88	3.60	.060
Example	18.41	3.92	.051
Text*Example	14.00	2.99	.087
Error	4.69		

These null results--suggesting that across conditions participants were not sorting structurally--are not surprising given that participants had only one lesson for programming in App Inventor. In the Chi et al. (1981) study, the novices were students who had completed a semester of college-level physics, and they still sorted problems on superficial features. It could be the case that even if subgoal labels helped effectively organize information, learners need more exposure to the domain knowledge to build robust, structurally-oriented mental representations for a topic. Additionally, participants might have been mentally fatigued by the time they received this task and overwhelmed by receiving 30 cards at once leading to poorly planned categorizations.

3.2.4 Generalization Task

The generalization task asked participants to describe in general terms how they would make an app given certain specifications. This assessment task was meant to measure how well they could create a high level description of the procedure for making an app. To score the generalization task, participants received a point for each structural feature that they described that was necessary for creating the app. Participants did not receive points for specific descriptions or unnecessary features. Specific descriptions included information about how to achieve a step using the interface or specified a particular block to be used. The maximum score on this assessment was 6. The ICC(A) for this assessment was .89.

In this assessment, there was a main effect of text design: people who received subgoal labeled text (*M* = 4.4, *SD* = 1.1) performed better than those who did not (*M* = 3.5, *SD* = 1.3), *F* (1, 116) = 15.11, *MSE* = 1.49, *p* < .001, est. ω^2 = .10, *f* = .35. There was no main effect of example design, *F* (1, 116) = 2.70, *MSE* = 1.49, *p* = .10, and there was no interaction, *F* (1, 116) = .20, *MSE* = 1.49, *p* = .66. The results indicate that receiving subgoal labels in the text improved performance, but receiving subgoal labels in the example did not.

There was also a main effect of text design for time on task in this assessment. Participants who received subgoal labeled text (*M* = 3.9, *SD* = 2.3) took longer to complete the task than those who did not (*M* = 3.0, *SD* = 1.20), *F* (1, 116) = 5.95, *MSE* =

3.48, p = .016, est ω^2 = .05, f = .22. There was no main effect for example design, F (1, 116) = .83, MSE = 3.48, p = .36, and there was no interaction of text and example design, F (1, 116) = 1.65, MSE = 3.48, p = .20. In this case, people who performed better took longer to complete the task.

Because responses were written, it could be the case that those who performed well needed more time to write their responses than those who performed poorly. To explore this possibility, the number of words in each response was counted to estimate the time participants spent writing. Participants who received subgoals in text wrote on average 40 words, whereas other participants wrote on average 34 words. This finding suggests that part of the difference in time on task between these groups is due to time spent writing.

CHAPTER 4

CONCLUSIONS

The present research advances knowledge about strategies for improving novice problem solving. Kirschner, Sweller, and Clark (2006) argued that guided instruction is important for novices because it provides them with an organizational structure to help store new information. The present study explored a new type of guided instruction: subgoal labeled instructional text.

The results from the problem solving, explanation, and generalization tasks suggest that learners would benefit most from receiving subgoal labels in both instructional text and worked examples. The results also suggest that subgoal labeled text help learners in a different way than the subgoal labeled examples. Both the generalization task and labeling portion of the explanation task required participants to articulate their knowledge of the general procedure. In both cases, participants who received subgoal labels in text outperformed those who did not.

To speculate on how subgoal labeled instructional materials affect learning, the model in Figure 1 that describes how subgoal labeled worked examples improve problem solving was expanded. Figure 8 shows a proposed model for how subgoal labels in text and examples jointly improve problem solving. The results of the present study provide some preliminary evidence to support the benefits of subgoal labeled instructional text and the combination of subgoal labels in both text and examples for problem solving. Evidence from other researchers supports some of the connections in this proposed model. Chi (2009) and Hill and Levenhagen (1995) support that self-explanations improve articulation and that improved articulation can lead to improved problem solving. Additionally, Eiriksdottir and Catrambone (2011) argue that integrating information from general and specific instructions can improve the application of general knowledge and transfer of specific knowledge. More research is needed, of course, to systematically test this model.

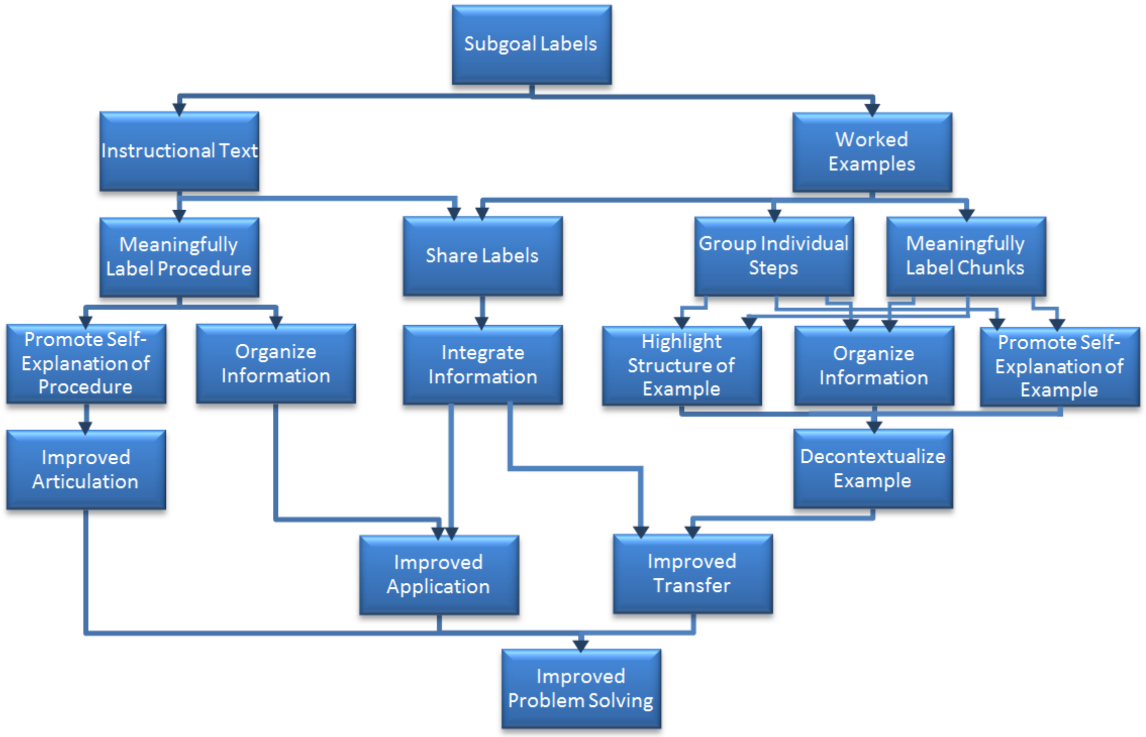


Figure 8. Diagram of how subgoal labeled instructional text and worked examples can help learners improve problem solving performance. Starting from the top, the first two levels describe the type of instructional material. The third level describes the physical characteristics of subgoal labels. The fourth level describes how the physical characteristics help the learners use effective learning strategies. The fifth and sixth levels describe how these strategies help learners understand the instructional material, and the last level describes the outcome.

Future research can also access the generalizability of these findings to other STEM instructions and for more advanced learners. Because most STEM subjects use the same approach to instruction (i.e., general instructional text and specific worked examples; CDSL, 2000), the subgoal labeled instructional materials can be adapted for and tested in other STEM subjects. Subgoals can also be scaled hierarchically to the knowledge level of the learner. For example, novices, who have little knowledge in the domain, would need the most basic or low-level subgoals for solving a problem, whereas more advanced learners could utilize higher-level subgoals that subsume multiple basic subgoals. As learners gain knowledge, higher-level processes become lower-level processes (CDSL, 2000) creating a need to scale subgoals based on the level of knowledge.

The subgoal intervention manipulates the instructional materials that students receive; therefore, reaching a large number of students with the work of a relatively small group of people (i.e., the instructional designers) would be relatively easy. Instructors would likely not need to be retrained because, as experts in the topics that they teach, they likely are familiar with the subgoals, though they might not have articulated specific labels for them. But as experts, the instructors might not recognize the support that novices need to effectively organize information (CDSL, 2000). Subgoal labeled instructional materials could supplement interpersonal instruction and help fill in the gaps in learning if instructors are unsuccessful in teaching these aspects of the information.

Because these interventions are not dependent on instructors, they can also be used in learning environments without personal interaction with an instructor, such as online learning. Though this study does not claim that students who study with these instructional materials alone would perform similarly to students who received these materials and instruction from an instructor, the study was conducted in a computer-based learning environment without an instructor. Therefore, the results of the study represent the results that could be expected if students used only these instructions.

Subgoal labeled worked examples have already been shown to significantly increase learners' problem solving performance (e.g., Catrambone, 1998). The present study demonstrated that subgoal labeled instructional text can increase this effect and improve articulation of procedures. The study suggests that subgoal labels should be used in both instructional text and worked examples used to teach problem solving procedures.

APPENDIX A

SUBGOAL LABELED INSTRUCTIONAL TEXT

In this session you will create an app that shows a picture of a fortune teller in a button. When you click the button, your fortune will be displayed. The fortune will be picked randomly from a list of possible fortunes.

To create the app, you'll use two different components of App Inventor.

In the App Inventor Designer

This is the first screen that comes up when you start a new project, and this is where you will set up the components of your app.

Create components

Components are the pieces that provide your app functionality, such as a *button* that users can press or a *label* to display information. You'll create components in the App Inventor Designer by selecting which type of component you want to create and dragging it to the screen. The components are on the left of the screen and are organized under different "palettes" which each have a theme (e.g., media or animation).

Set properties

You'll be able to change the properties of each component in the App Inventor Designer as well. For example, you can change how big a *button* is or change the font of a *label* to bold in the "Properties" section on the right of the screen. The properties that can be set depend on the component that is being manipulated.

In the App Inventor Blocks Editor

The Blocks Editor is opened by click on the "Open the Blocks Editor" button in the Designer, and this is where you will program the components of your app.

Handle events from My Blocks

Blocks are the user and computer actions that you'll piece together to program your app. My Blocks is the section of blocks that contains the blocks for the components of your app; that is, if you create a *button*, then the blocks for the *button* will be in My Blocks. To program a feature of your app, you'll first need to define which input, from the user or computer, will start the program. These inputs will almost always come from My Blocks. For example, if you want to create a feature, so text is displayed when a button is clicked, you’ll need to start with the block “when button.click,” so the program knows after what action to display the text.

Set outputs from My Blocks

Similarly, to programming the feature, you’ll also need to define what output you want. These outputs will almost always come from My Blocks. From the previous example, if you want to display the text on a label, then you’ll need to add the block “set label.text” to the “when button.click” block.

Set conditions from Built-in

The Built-in blocks are blocks that are not dependent on which components your app has. Built-in blocks allow you to add features, such as variables, to your app with which the user will not directly interact. You can use these blocks to create conditions for your program. From the previous example, if you wanted the program to randomly select the text to be displayed from a list of text items, then you’d need to create a list and add the “call select list item” block to the “set label.text” block.

Define variables from Built-in

Variables are a value that can be changed. By defining a variable, you are giving that value a name that can be used in a program. From the previous example, the text that is displayed from the list of text items is a variable. Because the text that is displayed can change, you’ll need to attach the variable block to the “call select list item” block.

The following video will demonstrate how to use Android App Inventor and show you how to make the Fortune Teller app.

At this time, please watch the video by clicking on the Media Player icon at the bottom of your screen. Make sure that you wear headphones while watching the video. This video will demonstrate how to create this app.

When you’re done watching the video, use the following steps to create your own Fortune Teller app.

APPENDIX B

UNLABELED INSTRUCTIONAL TEXT

In this session you will create an app that shows a picture of a fortune teller in a button. When you click the button, your fortune will be displayed. The fortune will be picked randomly from a list of possible fortunes.

To create the app, you'll use two different components of App Inventor.

In the App Inventor Designer

This is the first screen that comes up when you start a new project, and this is where you will set up the components of your app.

Components are the pieces that provide your app functionality, such as a *button* that users can press or a *label* to display information. You'll create components in the App Inventor Designer by selecting which type of component you want to create and dragging it to the screen. The components are on the left of the screen and are organized under different "palettes" which each have a theme (e.g., media or animation).

You'll be able to change the properties of each component in the App Inventor Designer as well. For example, you can change how big a *button* is or change the font of a *label* to bold in the "Properties" section on the right of the screen. The properties that can be set depend on the component that is being manipulated.

In the App Inventor Blocks Editor

The Blocks Editor is opened by click on the "Open the Blocks Editor" button in the Designer, and this is where you will program the components of your app.

Blocks are the user and computer actions that you'll piece together to program your app. My Blocks is the section of blocks that contains the blocks for the components of your app; that is, if you create a *button*, then the blocks for the *button* will be in My Blocks. To program a feature of your app, you'll first need to define which input, from the user or computer, will start the program. These inputs will almost always come from My Blocks. For example, if you want to create a feature, so text is displayed when a button is clicked, you'll need to start with the block "when button.click," so the program knows after what action to display the text.

Similarly, to programming the feature, you'll also need to define what output you want. These outputs will almost always come from My Blocks. From the previous example, if you want to display the text on a label, then you'll need to add the block "set label.text" to the "when button.click" block.

The Built-in blocks are blocks that are not dependent on which components your app has. Built-in blocks allow you to add features, such as variables, to your app with which the user will not directly interact. You can use these blocks to create conditions for your program. From the previous example, if you wanted the program to randomly select the text to be displayed from a list of text items, then you'd need to create a list and add the "call select list item" block to the "set label.text" block.

Variables are a value that can be changed. By defining a variable, you are giving that value a name that can be used in a program. From the previous example, the text that is displayed from the list of text items is a variable. Because the text that is displayed can change, you'll need to attach the variable block to the "call select list item" block.

The following video will demonstrate how to use Android App Inventor and show you how to make the Fortune Teller app.

At this time, please watch the video by clicking on the Media Player icon at the bottom of your screen. Make sure that you wear headphones while watching the video. This video will demonstrate how to create this app.

When you're done watching the video, use the following steps to create your own Fortune Teller app.

APPENDIX C

SUBGOAL LABELED STEP-BY-STEP GUIDE

1. Go to the Android App Inventor website by clicking the Firefox icon that is on the bottom of your screen.
2. Create a new project by clicking on *New* and naming the project "fortune" and your participant number (e.g., “fortune1”). Ask the moderator if you do not know your participant number.

In the Designer

Create Component

3. From the basic palette drag out a *Button*.

   Buttons are components that users touch to perform some action in your app. Buttons detect when users tap them. Many aspects of a button's appearance can be changed. You can use the Enabled property to choose whether a button can be tapped.

Set Properties

4. Set the image source to "gypsy.jpg". This file will be located in the “Media” folder on the desktop.
5. Clear the default text.
6. Set the width to fill the parent's width and the height to 300 pixels.

Create Component

7. From the basic palette drag out a *Label*.

   Labels are components used to show text. A label displays text which is specified by the Text property. Other properties, all of which can be set in the Designer or Blocks Editor, control the appearance and placement of the text.
8. Place the *Label* underneath the gypsy image.

Set Properties

9. Set the text to "Click button to see your fortune".
10. Rename it to "fortuneLabel".

In the Blocks Editor

11. Open the blocks editor.

Define Variables from Built-in

12. Click on "Built-In" and "Definition" and pull out a *def variable.*

A ***variable*** creates a value that can be changed while an app is running, and gives that value a name. Variables are global in scope, which means you can refer to them from any code in the app, including from within procedures.

When you create a variable, App Inventor will automatically create two associated blocks, and place them in the My Definitions drawer in My Blocks:

- The global block gets the value of the variable.
- The set global block changes the value of the variable.

13. Click on the "variable" and replace it with "fortuneList". This creates a variable called "fortuneList".
14. Click on "Lists" and drag out a *call make a list*

Make a list creates a list from the given blocks. If you don't supply any arguments, this creates an empty list, which you can add elements to later.

15. Click on "Text" and drag out a *text text* block and drop it next to "item".

Text contains a text string.

16. Click on the rightmost "text" and replace it with your first fortune.
17. Repeat steps 15 and 16 to create 3 additional fortunes.

Handle Events from My Blocks

18. Click on "My Blocks" and "Button1".
19. Drag out a *when Button1.Click.*

Set Output from My Blocks

20. Click on "fortuneLabel"
21. Drag out a *set fortuneLabel.Text to* and drop it in the *when Button1.Click*

Set Conditions from Built-in

22. Click on "Built-In" and "Lists"
23. Drag out a *call select list item*

Select list item selects the item at the given index in the given list.

24. Click on "My Blocks" and "My Definitions"
25. Drag out a *global fortuneList* and put that next to the area marked "list".
26. Click on "Built-In" and "Math"
27. Drag out a *call random integer* and drop it in the area marked "index".

Random integer returns a random integer value between the given values, inclusive. The order of the arguments doesn't matter.

28. Remove the "100" number block next to the "to" area by throwing it in the trash.

29. Click on "Lists"

30. Drag out a *call length of list* and drop it in the "to" area.

    Length of list returns the number of items in the list.

31. Click on "My Blocks" and "My Definitions"

32. Drag out a *global fortuneList* and drop it after the area marked "list" in *call length of list*.

Now you have a Fortune Teller app!

APPENDIX D

UNLABELED STEP-BY-STEP GUIDE

1. Go to the Android App Inventor website by clicking the Firefox icon that is on the bottom of your screen.
2. Create a new project by clicking on *New* and naming the project "fortune" and your participant number (e.g., “fortune1”). Ask the moderator if you do not know your participant number.
3. From the basic palette drag out a *Button*.

   Buttons are components that users touch to perform some action in your app. Buttons detect when users tap them. Many aspects of a button's appearance can be changed. You can use the Enabled property to choose whether a button can be tapped.
4. Set the image source to "gypsy.jpg". This file will be located in the “Media” folder on the desktop.
5. Clear the default text.
6. Set the width to fill the parent's width and the height to 300 pixels.
7. From the basic palette drag out a *Label*.

   Labels are components used to show text. A label displays text which is specified by the Text property. Other properties, all of which can be set in the Designer or Blocks Editor, control the appearance and placement of the text.
8. Place the *Label* underneath the gypsy image.
9. Set the text to "Click button to see your fortune".
10. Rename it to "fortuneLabel".
11. Open the blocks editor.
12. Click on "Built-In" and "Definition" and pull out a *def variable.*

    A ***variable*** creates a value that can be changed while an app is running, and gives that value a name. Variables are global in scope, which means you can refer to them from any code in the app, including from within procedures.

    When you create a variable, App Inventor will automatically create two associated blocks, and place them in the My Definitions drawer in My Blocks:

    - The global block gets the value of the variable.
    - The set global block changes the value of the variable.

13. Click on the "variable" and replace it with "fortuneList". This creates a variable called "fortuneList".

14. Click on "Lists" and drag out a *call make a list*

    Make a list creates a list from the given blocks. If you don't supply any arguments, this creates an empty list, which you can add elements to later.

15. Click on "Text" and drag out a *text text* block and drop it next to "item".

    Text contains a text string.

16. Click on the rightmost "text" and replace it with your first fortune.

17. Repeat steps 15 and 16 to create 3 additional fortunes.

18. Click on "My Blocks" and "Button1".

19. Drag out a *when Button1.Click*.

20. Click on "fortuneLabel"

21. Drag out a *set fortuneLabel.Text to* and drop it in the *when Button1.Click*

22. Click on "Built-In" and "Lists"

23. Drag out a *call select list item*

    Select list item selects the item at the given index in the given list.

24. Click on "My Blocks" and "My Definitions"

25. Drag out a *global fortuneList* and put that next to the area marked "list".

26. Click on "Built-In" and "Math"

27. Drag out a *call random integer* and drop it in the area marked "index".

    Random integer returns a random integer value between the given values, inclusive. The order of the arguments doesn't matter.

28. Remove the "100" number block next to the "to" area by throwing it in the trash.

29. Click on "Lists"

30. Drag out a *call length of list* and drop it in the "to" area.

    Length of list returns the number of items in the list.

31. Click on "My Blocks" and "My Definitions"

32. Drag out a *global fortuneList* and drop it after the area marked "list" in *call length of list*.

Now you have a Fortune Teller app!

APPENDIX E

PROBLEM SOLVING TASK

Write the steps you would take to italicize the fortune presented.

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).

Write the steps you would take to create a list of colors and make the ball change to a random color whenever it collided with something.

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.

APPENDIX F

EXPLANATION TASK

The sheet you received has the steps to the solutions of the problems that you were just working on. The steps are correct and in the correct order. Please group the steps of these solutions that you think go together (either by circling them or drawing a bracket around them). “Go together” is open to your interpretation, but think of it as if you were trying to put headers into the solution to group steps in some meaningful way. If you do not think any of the steps go together, you do not have to group any steps. If you group steps together, please provide a label or description of why you think those steps go together.

Write the steps you would take to italicize the fortune presented.

Select Label
Click “FontItalic”

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).

Drag out Ball
Set Heading to 0
Set Interval to 250
Set Speed to 5

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.

```
Drag out "def variable"
Add "call make a list" and put it in "as"
Add colors to list
Drag out "when Ball1.CollidedWith"
Add "set Ball1.PaintColor" and put it in "do"
Add "call select list item" and put it in "do"
    Add "global color" and put it in "list"
    Add "call random integer" and put it in "index"
        Delete "100" from "to"
        Add "call length of list" and put it in "to"
            Add "global color" and put it in "list"
```

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.

```
Drag out "when Ball1.Touched"
Add "set Ball1.Heading" and put in "do"
From math, add "+" block
Add "Ball1.Heading" and "90" to the "+" block
```

APPENDIX G

INSTRUCTIONS FOR CARD SORTING TASK

The third assessment is a card sorting task. You’ll get 30 cards that have parts of apps on them. The task is to **group cards that go together** and then **label each group**.

You can make 2 to 6 groups, but do not make more than 6 groups.

To create a group of cards, you can select the cards that you think should go together, then right click and select “Add to New Group” from the menu that pops up. You can also create groups by clicking the “Add Group” button on the top left and put cards in the group by ctrl key + left click on the card and drag it into the group. You can also use ctrl + click to move cards to different groups.

To label the group, double click on the title and type in a label describing the purpose of the cards in that group.

APPENDIX H

CARDS IN CARD SORTING TASK GROUPED BY SUBGOAL

Create Components

1. From the basic palette drag out a Canvas
2. From the animation palette drag out a Ball
3. From the Basic palette drag out a Button
4. From the basic palette drag out a Label
5. From the animation palette drag out an ImageSprite
6. From the media palette drag out a Sound

Set Properties

7. Set the width of the canvas to fill the parent`s width and the height to 390 pixels
8. Set the ball`s radius to 20 and rename it to "ball"
9. Set the width to fill the parent`s width and the height to 300 pixels. Set the text for the button to "Push Me!". Set the font size to 36. Set the background color to pink.
10. Set the text of the label to "Score:"
11. Change the name of the Image Sprite to "clap" and set the image file to "hand_clap.gif"
12. Change the name of the sound to "clapSound" and set the source file to "clap.wav"

Handle Events from My Blocks

13. Click on "startButton" and drag out a when startButton.Click block
14. From "paddle" drag out a when paddle.Dragged block
15. Click on "pushButton" and drag out a "when pushButton.Click" block
16. Click on "MyBlocks" and then on "timer" and drag out a "when timer.Timer"
17. Click on "clap" and drag out a when clap.Touched block
18. Click on "AccelerometerSensor1" and drag out when AccelerometerSensor1.AccelerationChanged

Set Outputs from My Blocks

19. Click on "ball" and drag out a set ball.PaintColor to block and put it inside the last block. Then, click in the empty space next to the block and click on "Colors" and "Green"
20. From "ball" drag out a set ball.Speed to block and drop it after the last block. Click next to the "to" and type 5
21. Click on "My Definitions" and drag out a "set global score to" block. Then, place this block in the "when pushButton.Click" block

22. Click on "My Definitions" and drag out a "set global time to" and put it inside the when timer.Timer block
23. Click on "clapSound" and drag out a call clapSound.Play and connect it after the when clap.Touched
24. Click on "drum2Sound" and drag out call drum2Sound.Play and put it next to the "then-do" part of the if block

Set Conditions from Built-in

25. From "Control" drag out an ifelse. Click on "Math" and drag out a blank = blank and drop it after "test". Drag out a value edge block and drop it in the first blank and add -1 to the second blank
26. Drag out an if and drop it after the call updateScore. Drag out a blank = blank and drop it in "test". Drag out a global score and drop it in the first blank, and put 10 in the second blank
27. Click on "Math" and drag out a "blank + blank" block. Drop this block after "set global score to". Drag out a "global score" block and drop it in the first blank area, and put 1 in the second blank
28. Click on "Math" and drag out a "blank - blank". Click on "My Definitions" and drag out a "global time" block and put it in the first blank, and put a 1 in the second blank
29. From “control” drag out an if and put it inside the block. From “math” drag out blank > blank and add it next to test. Put AccelerometerSensor1.XAccel in first blank and 3 in the second blank
30. From “Control” drag out an if and put it inside the block. From “math” drag out blank > blank and add it next to test. Put AccelerometerSensor1.YAccel in first blank and -3 in the second blank

APPENDIX I

GENERALIZATION TASK

Describe the general procedure you would take to create an app that has an image and a sound, so that the sound played when the image was touched. You do NOT need to list the specific steps, just the general procedure.

A good first step would be, “Make a component for the image.”

A bad first step would be, “Drag an image sprite from the palette to the canvas,” because it’s too specific.

REFERENCES

Atkinson, R. K., Catrambone, R., & Merrill, M. M. (2003). Aiding transfer in statistics: Examining the use of conceptually oriented equations and elaborations during subgoal learning. Journal of Educational Psychology, 95(4), 762-773. doi: 10.1037/0022-0663.95.4.762

Atkinson, R. K., & Derry, S. (2000). Computer-based examples designed to encourage optimal example processing: A study examining the impact of sequentially presented, subgoal-oriented worked examples. In B. Fishman & S. O'Connor-Divelbiss (Eds.), Fourth International Conference of the Learning Sciences, (pp. 132-133), Mahwah, NJ: Earlbaum.

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: 10.2307/1170661

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

Bassok, M., & Holyoak, K. (1989). Interdomain transfer between isomorphic topics in algebra and physics. Journal of Experimental Psychology: Learning, Memory, and Cognition, 15(1), 153-166. doi: 10.1037/0278-7393.15.1.153

Beatty, A. (Rapporteur), Committee on Highly Successful Schools or Programs for K-12 STEM Education, National Research Council. (2011). Successful STEM education: A workshop summary. Retrieved from http://www.nap. edu/catalog. php?record_id=13230

Biederman, I., & Shiffrar, M. (1987). Sexing day-old chicks: A case study and expert systems analysis of a difficult perceptual-learning task. Journal of Experimental Psychology: Learning, Memory, and Cognition, 13(4), 640-645. doi: 10.1037/0278-7393.13.4.640

Catrambone, R. (1990). Specific versus general procedures in instructions. Human-Computer Interaction, 5(1), 49-93. doi: 10.1207/s15327051hci0501_2

Catrambone, R. (1994). Improving examples to improve transfer to novel problems. Memory and Cognition, 22, 605-615. doi: 10.3758/BF03198399

Catrambone, R. (1995a). Aiding subgoal learning: Effects on transfer. Journal of Educational Psychology, 87(1), 5-17. doi: 10.1037/0022-0663.87.1.5

Catrambone, R. (1995b). Following instructions: Effects of principles and examples. Journal of Experimental Psychology: Applied, 1(3), 227-244. doi: 10.1037/1076-898X.1.3.227

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. (2013). Task Analysis by Problem Solving (TAPS): A method for uncovering expert knowledge. Unpublished manuscript, School of Psychology, Georgia Institute of Technology, Atlanta, GA.

Catrambone, R., & Holyoak, K. (1990). Learning subgoals and methods for solving probability problems. Memory & Cognition, 18(6), 593-603. doi: 10.1037/0096-3445.127.4.355

Chi. M. T. H. (2009). Active-constructive-interactive: A conceptual framework for differentiating learning activities. Topics in Cognitive Science, 1(1), 73-105.

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., Leeuw, N. D., Chiu, M., LaVancher, C. (1994). Eliciting self-explanations improves understanding. Cognitive Science, 18¸439-477.

Committee on Developments in the Science of Learning, National Research Council. (2000). How people learn: Brain, mind, experience, and school: Expanded edition. Retrieved from http://www.nap. edu/catalog.php?record_id=9853

Committee on Highly Successful Schools or Programs in K-12 STEM Education, National Research Council. (2011). Successful K-12 STEM education: Identifying effective approaches in science, technology, engineering, and mathematics. Retrieved from http://www.nap.edu/catalog.php?record_id=13158

Eiriksdottir, E., & Catrambone, R. (2011). Procedural instructions, principles, and examples: How to structure instructions for procedural tasks to enhance performance, learning, and transfer. Human Factors, 53(6), 749-770. doi: 10.1177/0018720811419154

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

Faul, F., Erdfelder, E., Lang, A.-G., & Buchner, A. (2007). G*Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. Behavior Research Methods, 39, 175-191.

Ferguson, A. N., Kazi, S., & Durso, F. T. (2012). Revealing latent strategy structures from expert critical care nurses [CD]. Proceedings of the Human Factors and Ergonomics Society 56th Annual Meeting (pp. 801-805).

Hill, R. C., & Levenhagen, M. (1995). Metaphors and mental models: Sensemaking and sensegiving in innovative and entrepreneurial activities. Journal of Management, 21(6), 1057-1074.

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

Katehi, L., Pearson, G., & Feder, M. (Eds.), Committee on K-12 Engineering Education, National Academy of Engineering and National Research Council. (2009). Engineering in K-12 education: Understanding the status and improving the prospects. Retrieved from http://www. nap.edu/catalog.php?record_id=12635

Kirschner, P., Sweller, J., & Clark, R. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational Psychologist, 41(2), 75-86. doi: 10.1207/s15326985ep4102_1

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

Margulieux, L. E., Guzdial, M., & Catrambone, R. (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 (pp. 71-78). New York, NY: Association for Computing Machinery. doi: 10.1145/2361276.2361291

Meyer, B. J. F., & McConkie, G. W. (1973). What is recalled after hearing a passage? Journal of Educational Psychology, 65, 109-117. doi: 10.1037/h0034762

Nielsen, N. (Rapporteur), Planning Committee on Evidence on Selected Innovations in Undergraduate STEM Education, National Research Council. (2011). Promising practices in undergraduate science, technology, engineering, and mathematics education: Summary of two workshops. Retrieved from http://www.nap.edu/ catalog.php?record_id=13099

Paas, F., & van Gog, T. (2009). Principles for designing effective and efficient training of complex cognitive skills. In F. T. Durso (Ed.) Reviews of Human Factors and Ergonomics (Vol. 5), Santa Monica, CA: HFES, pp. 166-194.

Palmiter, S., Elkerton, J., & Baggett, P. (1991). Animated demonstrations versus written instructions for learning procedural tasks: A preliminary investigation. International Journal of Man-Machine Studies, 34, 687-701. doi: 10.1016/0020-7373(91)90019-4

Reder, L. M., & Anderson, J. R. (1980). A comparison of texts and their summaries: Memorial consequences. Journal of Verbal Learning and Verbal Behavior, 19, 121-134. doi: 10.1016/S0022-5371(80)90122-X

Renkl, A. (2002). Worked-out examples: Instructional explanations support learning by self-explanations. Learning and Instruction, 12, 529-556. doi: 10.1016/S0959-4752(01)00030-5

Renkl, A., & Atkinson, R. K. (2002). Learning from examples: Fostering self-explanations in computer-based learning environments. Interactive Learning Environments, 10(2), 105-199. doi: 10.1076/ilee.10.2.105.7441

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.

Smith, E. E., & Goodman, L. (1984). Understanding written instructions: The role of an explanatory schema. Cognition and Instruction, 1, 359-396. doi: 10.1207 /s1532690xci0104_1

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.

VanLehn, K., Jones, R., & Chi, M. T. H. (1992). A model of the self-explanation effect. The Journal of the Learning Sciences, 2, 1-59. doi: 10.1207/s15327809jls0201_1