Board 37: Developing Subgoal Labels for Imperative Programming to Improve Student Learning Outcomes

Dr. Adrienne Decker, University at Buffalo

Adrienne Decker is a faculty member in the newly formed Department of Engineering Education at the University at Buffalo. She has been studying computing education and teaching for over 15 years, and is interested in broadening participation, evaluating the effectiveness of pre-college computing activities, and issues of assessment, particularly in the introductory programming courses. She has been actively involved with the Advanced Placement Computer Science A course since 2011, first serving as a reader, and as part of the development committee for the exam since 2015, serving as higher ed co-chair since 2018. She has received more than $1M in NSF funding for her work in computing education. Active in the computing education community, she is currently the ACM Special Interest Group on Computer Science Education board treasurer (2016-2019) and has served as program co-chair in 2014 and symposium co-chair in 2015 to the SIGCSE Technical Symposium on Computer Science Education.

Dr. Briana Morrison, University of Nebraska Omaha

Briana Morrison is an Assistant Professor at the University of Nebraska Omaha. Prior to joining the college of IS&T, Briana worked for IBM for 8 years as a software developer and then transitioned to academia. She was an Assistant Professor at Southern Polytechnic State University (now Kennesaw State University) for 20 years in the Computer Science department. She was the Undergraduate Coordinator for the Computer Science and Software Engineering programs, helped to found the Computer Game Design and Development degree program, and served as the lead for 2 successful ABET accreditation visits. She has a PhD in Human-Centered Computing from the Georgia Institute of Technology, a master's in Computer Science, and a bachelor's degree in Computer Engineering. Her research area is Computer Science Education where she explores cognitive load theory within programming, broadening participation in computing and expanding and preparing computing high school teachers.

Dr. Lauren Elizabeth Margulieux, Georgia State University

Lauren Margulieux is an Assistant Professor of Learning Sciences at Georgia State University. She received her Ph.D. from Georgia Tech in Engineering Psychology, the study of how humans interact with technology. Her research interests are in educational technology and online learning, particularly for computing education. She focuses on designing instructions in a way that supports online students who do not necessarily have immediate access to a teacher or instructor to ask questions or overcome problem solving impasses.

Developing Subgoal Labels for Imperative Programming to Improve Student Learning Outcomes

Overview

This NSF IUSE project incorporates instructional materials and techniques into introductory programming identified through educational psychology research as effective ways to improve student learning and retention. The research team has developed worked examples of problems that incorporate subgoal labels, which are explanations that describe the function of steps in the problem solution to the learner and highlight the problem solving process. Using subgoal labels within worked examples, which has been shown effective in other STEM fields, is intended to break down problem solving procedures into pieces that are small enough for novices to grasp. Experts, including instructors, teaching introductory level courses are often unable to explain the subgoal-level processes that they use in problem solving because they have automated much of the problem solving processes after many years of practice. This intervention had been tested in programming for a few hours of instruction and found effective. The current project expands upon that work.

The overarching research questions for this project are as follows:

1. How do subgoal-labeled worked examples affect learning through an entire introductory programming course?
2. How can formative assessments that are subgoal labeled impact student learning?

In order to answer these questions, we developed the following project goals:

- Develop a set of worked examples segmented by subgoal labels for an introductory programming course taught using an imperative programming language.
- Empirically test the worked examples in classrooms at multiple institutions with students of various backgrounds and majors.
- Develop a set of practice problems (formative assessments) that include subgoal labels that correspond to those in the worked examples.
- Empirically test assessment questions that include subgoal labels.

Background

There have been many calls recently for computing for all students across the nation. While there are many opportunities to study and use computing to advance the fields of computer science, software development, and information technology, computing is also needed in a wide range of other disciplines, including engineering. Most engineering programs require students take a course that teaches them introductory programming, which covers many of the same topics as an introductory course for computing majors (and at times may be the same course). However, statistics about the success of a course that is an introductory programming course are sobering; approximately half the students will fail, forcing them to either repeat the course or leave their chosen field of study if passing the course is required [1, 2].

We aim to improve success in introductory programming courses by incorporating subgoal learning. Subgoal learning explicitly teaches students the subgoals, or functional pieces, of a problem solving procedure, which can help novices in the problem solving process because they often do not recognize these functional pieces on their own [3]. It is often the case that subgoal labels are used in conjunction with worked examples. Worked examples are commonly used to teach problem solving procedures for well-structured problems because they demonstrate how to apply an abstract procedure to a concrete problem before the learner can solve problems independently [4-6]. Several studies have shown that subgoal learning in introductory programming courses has improved novice performance [7-11].

Creation of subgoals

The first part of our work involved determining a topic list for the subgoal label tasks that is representative of the topics that are commonly taught in introductory courses. Through experience in teaching introductory programming along with reviewing several best-selling textbooks, we determined this list to be:

- Assignment
- Selection
- Repetition (both definite and indefinite)
- Procedure / method writing and invocation (parameter passing)
- Object usage and class implementation (for object-oriented courses)
- Array processing

Next, we used the Task Analysis by Problem Solving (TAPS) protocol developed by Catrambone to identify the subgoals of the procedures [12]. Figure 1 lists the subgoal labels that were developed. Following identification, worked examples and practice problems of varying difficulty were created for each of the topics and integrated throughout an introductory programming course.

Results and observations of deployment in Fall 2018

The subgoals were deployed in Fall 2018 in the introductory programming courses at University of Nebraska Omaha. There were five sections of the course taught in Fall 2018, three used traditional materials (i.e., worked examples and formative assessments), and two sections used the subgoal labeled materials. The course, for all sections, was designed as a flipped class in which students watched recorded lectures before class and then answered peer instruction questions during class and problem solved in small groups. There were common quizzes among the sections of the course given throughout the term and data has been collected about student performance between the two groups.

Results of this preliminary analysis are detailed in [13] and suggest that the subgoal materials helped learners to solve problems using the procedure more effectively during the early stages of learning. Subgoal learners were also more likely to submit all of the exams which could mean

that the subgoal materials helped students who would have dropped out of the course to perform well enough to persist in the course. More analysis of the data is needed to determine any further conclusions.

Future Work

The next phase of this work involves a larger scale deployment for academic year 2019-2020 and data collection from several pilot sites. Recruitment will be happening through May 2019 and training and course materials will be made available to the teachers in July 2019.

Figure 1. Subgoals piloted in Fall 2018.

Subgoals for evaluating and writing expression (assignment) statements	
A. Evaluate expression statement	B. Write expression statement
1. Determine whether data type of expression is compatible with data type of variable 2. Update variable for pre based on side effect 3. Solve arithmetic equation 4. Check data type of copied value against data type of variable 5. Update variable for post based on side effect	1. Determine expression that will yield variable 2. Determine data type and name of variable and data type of expression 3. Determine arithmetic equation with operators 4. Determine expression components 5. Operators and operands must be compatible
Subgoals for evaluating and writing selection statements	
A. Evaluate selection statement	B. Write selection statement
1. Diagram which statements go together 2. For if statement, determine whether expression is true or false 3. If true – follow true branch, if false –follow else branch or do nothing if no else branch	1. Define how many mutually exclusive paths are needed 2. Order from most restrictive/selective group to least restrictive 3. Write if statement with Boolean expression 4. Follow with true bracket including action 5. Follow with else bracket 6. Repeat until all groups and actions are accounted for
Subgoals for evaluating and writing loops.	
A. Evaluate loops	B. Write loops
1. Identify loop parts a. Determine start condition b. Determine update condition c. Determine termination condition d. Determine body that is repeated 2. Trace the loop a. For every iteration of loop, write down values	1. Determine purpose of loop a. Pick a loop structure (while, for, do_while) 2. Define and initialize variables 3. Determine termination condition a. Invert termination condition to continuation condition 4. Write loop body a. Update loop control variable to reach termination
Subgoals for calling and writing methods	
A. Call or trace method calls	B. Write methods

1. Classify method as `static` method or `instance` method a. If `static`, use the class name b. If `instance`, must have or create an instance 2. Write (instance / class) dot method name and ( ) 3. Determine whether parameter(s) are appropriate a. Number of parameters passed must match method declaration b. Data types of parameters passed must match method declaration (or be assignable) 4. Determine what the method will return (if anything: data type, void, print, change state of object) and where it will be stored (nowhere, somewhere) 5. Evaluate right hand side of assignment (if there is one). Value is dependent on method's purpose	1. Define method header based on problem 2. Define return statement at the end 3. Define method body/logic a. Determine types of logic (expression, selection, loop, etc.) b. Define internal variables c. Write statements
Subgoals for using objects and writing classes	
A. Use objects (creating instances)	B. Write classes (associated rules sheet)
1. Declare variable of appropriate class datatype. 2. Assign to variable: keyword `new`, followed by class name, followed by (). 3. Determine whether parameter(s) are appropriate (API) a. Number of parameters b. Data types of the parameters	1. Name it 2. Differentiate class-level (`static`) vs. instance/object-level variables 3. Differentiate class-level (`static`) vs. instance/object behaviors/methods 4. Define instance variables (that you want to be interrelated) 5. Define class variables (`static`) as needed 6. Create constructor (behavior) that creates initial state of object 7. Create 1 accessor and 1 mutator behaviors per attribute 8. Write toString method 9. Write equals method 10. Create additional methods as needed
Subgoals for evaluating and writing arrays	
A. Evaluate arrays	B. Write arrays
1. Set up array from 0 to size-1 2. Evaluate data type of statements against array 3. Trace statements, updating slots as you go a. Remember assignment subgoals	1. Data type plus **[ ]** 2. Variable name = `{initializer list}`, or `new datatype [size]`

Acknowledgements

This material is based upon work supported by the U.S. National Science Foundation under Grant Nos. 1712025 and 1712231.

References

[1] Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, *39*(2), 32–36.
[2] Watson, C., Li, F. W., & Godwin, J. L. (2014). No tests required: comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM technical*

symposium on Computer science education (pp. 469–474). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=2538930
[3] Catrambone, R. (1998). The subgoal learning model: Creating better examples so that students can solve novel problems. Journal of Experimental Psychology: General, 127(4), 355.
[4] Atkinson, R. K., Derry, S. J., Renkl, A., & Wortham, D. (2000). Learning from examples: Instructional principles from the worked examples research. Review of Educational Research, 70(2), 181-214.
[5] Schwonke, R., Renkl, A., Krieg, C., Wittwer, J., Aleven, V., & Salden, R. (2009). The worked-example effect: Not an artefact of lousy control conditions. Computers in Human Behavior, 25(2), 258-266.
[6] Sweller, J. (2006). The worked example effect and human cognition. Learning and Instruction.
[7] Brown, N. C., & Wilson, G. (2018). Ten quick tips for teaching programming. PLoS Computational Biology, 14(4).
[8] Joentausta, J., & Hellas, A. (2018, February). Subgoal Labeled Worked Examples in K-3 Education. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (pp. 616-621). ACM.
[9] Margulieux, L. E., Guzdial, M., & Catrambone, R. (2012). Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In Proceedings of the Ninth Annual International Conference on International Computing Education Research (pp. 71-78). New York, NY: ACM.
[10] Morrison, B. B., Decker, A., & Margulieux, L. E. (2016). Learning loops: A replication study illuminates impact of HS courses. In Proceedings of the Twelfth Annual International Conference on International Computing Education Research (pp. 221-230). New York, NY: ACM.
[11] Morrison, B. B., Margulieux, L. E., & Guzdial, M. (2015). Subgoals, context, and worked examples in learning computing problem solving. In Proceedings of the Eleventh Annual International Conference on International Computing Education Research (pp. 21-29). New York, NY: ACM.
[12] Catrambone, R. (2011). Task analysis by problem solving (TAPS): Uncovering expert knowledge to develop high-quality instructional materials and training. Paper presented at the 2011 Learning and Technology Symposium (Columbus, GA, June).
[13] Margulieux, L. E., Morrison, B. B., and Decker, A. (2019). Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1. In Proceedings of the 24th Annual 24th Annual Conference on Innovation and Technology in Computer Science Education. New York, NY: ACM.