Pedagogy and Measurement of Program Planning Skills

Francisco Enrique Vicente G. Castro
Department of Computer Science
Worcester Polytechnic Institute
Worcester, Massachusetts 01609 USA
fgcastro@cs.wpi.edu

ABSTRACT

Students in first-year computing courses deal with programming problems that can be solved through various solutions that organize the problem's tasks in different ways. Selecting among these organizations of code or program *plans* and implementing them depends on various factors such as prior knowledge of solutions and features of programming languages used. Additionally, problem solving through effective program planning continues to be difficult for students; instruction in many first-year courses focuses on low-level constructs without discussing higher-level plans and students are left to figure out strategies for problem decomposition and code composition on their own. My research explores ways to teach planning strategies effectively so students will be able to learn to apply these strategies as well as transfer the use of these strategies across problems.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education—*Computer Science Education*.

Keywords

Program design, programming pedagogy, plan composition

1. PROGRAM CONTEXT

I am a full-time doctoral student and research assistant beginning the third year of my PhD program in Computer Science at WPI. I have gained candidacy by completing my research qualifying requirement and PhD breadth requirement in Spring 2016. My program coursework includes learning science courses to inform my research.

My research qualifier was an exploratory study of planning behavior of CS1 students; results of this study were also presented in a conference. I am doing follow-up studies that explore the effect of planning-focused pedagogical interventions on how students structure their solutions to programming problems and exploring how to measure planning skill acquisition and the transfer of these skills to other problems. The findings will contribute to the development of a dissertation proposal in the coming academic year.

ICER '16 September 08-12, 2016, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-4449-4/16/09.

DOI: http://dx.doi.org/10.1145/2960310.2960344

2. CONTEXT AND MOTIVATION

Students make various choices when developing solutions for programming problems. These choices range from lower-level concerns of selecting programming constructs to use (e.g. choosing between looping constructs) to higher-level concerns such as how to cluster the subtasks of a problem into functions or code blocks. The organization and clustering of subtasks is called a *plan* [7]. Consequently, programming is a process that involves both the implementation of plan components using lower-level constructs and the composition of these implementations into a solution that meets the goals of the programming problem.

Arguably, program planning is a useful skill, not only for students who study computing or intend to work in software development, but also for casual programmers who write code to, say, process data for lab experiments. These people encounter situations such as noisy data and changing process requirements that require planning and not just low-level construct choices. However, the landscape of first-year programming courses focuses instruction on the use of low-level programming constructs while expecting students to develop problem-solving and programming strategies on their own through extensive trial-and-error [2]. A recent study I have conducted showed that students struggle with programming even with the use of scaffolding code when there is no explicit instruction on program planning [1].

3. BACKGROUND & RELATED WORK

Developing and integrating programming plans has been identified as a difficult task among programming students [7]. While more recent studies have shown students succeed in program planning in specific contexts [4, 6], the pedagogic choices that help students with this task, as well as the prior knowledge that students bring with them remain poorly understood. Students who take their first computing courses in college vary in terms of programming background – some will carry some experience from high school while for others, this is their first exposure to programming.

Alongside the resurgence of planning studies [1, 6] is a growth in the body of research that addresses the improvement of planning skills through pedagogical frameworks and curricula that teach problem solving strategies in programming. Muller *et al.* used the concept of programming *patterns*, known solutions for recurring design or programming problems, in developing pattern-oriented instruction [5]. This approach teaches students to attach labels to algorithmic patterns and encourages them to look for common patterns across problems. De Raadt *et al.* used a 'strategy guide'

that discusses *abutment*, *nesting*, and *merging* for integrating programming strategies and explicitly required their students to apply specific strategies in their solutions [3]. My research builds on these by focusing on *problem-level techniques* (e.g. cleaning data) rather than *code-level techniques* (e.g. merging code). Additionally, I am exploring students' discussions on design tradeoffs as a way of surveying what they understand about patterns and their motivations for choosing between them.

4. STATEMENT OF THESIS/PROBLEM

My work explores how to make students effective at planning programs with multiple subtasks. Specifically, I want to develop (a) techniques for measuring planning and related skills in students, and (b) techniques for teaching these skills in ways that improve students' planning performance.

Based on preliminary findings and lessons learned in my previous studies, I propose that students who have acquired sufficient planning skills should be able to do the following:

1. **Planning:** apply planning in producing programs by decomposing the problem into relevant tasks, implementing the tasks in code, and composing the implementations into an overall program.
2. **Assessment:** provide a technically accurate assessment of plans using a vocabulary of planning terminology.
3. **Transfer:** apply planning to other problems requiring similar plans but in different contexts.

5. RESEARCH GOALS & METHODS

I will focus on the following concrete goals and methods:

Develop a framework for teaching planning. An integral part of this is exposing students not only to problems with multiple subtasks, but also to multiple plans for each problem. At this stage, I am building a library of problems that can be used for teaching and assessment and determining what techniques and principles to teach students for designing plans.

Measure the acquisition of planning skill. Measuring learning gains requires assessing students' prior knowledge. I will collect data on students' programming backgrounds and ability to write programs for simple problems. This will provide a baseline of what plans students know before instruction on planning in a first-year computing course. To measure acquired planning knowledge, students' solutions to assessment problems will be analyzed to determine (a) what plans students use, (b) ability to develop correct plans for problems, and (c) whether they can develop multiple plans for a problem - this last measure gives insight into students' skills in approaching a problem in multiple ways.

Measure planning assessment skill. Programming problems can be solved through multiple approaches that each have design tradeoffs. Good planning education enables students to produce multiple plans for each problem and appreciate tradeoffs between plans. I will collect and analyze data on issues and principles students raise when discussing programming solutions before and after instruction. Doing so allows me to determine whether students gain richer and more accurate planning vocabulary and whether being made aware of multiple solutions changes their plans when solving problems. This also aids in gaining insight into their problem solving knowledge and processes.

Measure transfer of planning skill. This enables the assessment of the transfer of learning in planning across problems. Multiple problems with different contexts but sharing common plans will be used for assessment to determine whether students are able to recognize similarities in plan requirements and are able to utilize their planning knowledge to solve these problems.

6. DISSERTATION STATUS

I have published an exploratory study [1] that looked at student programming behavior when working on problems for which they have not seen techniques for solving. Findings show that even when scaffolding is provided (i.e. template code for traversing input data), students struggled to develop working solutions. This shows that we clearly need to figure out richer approaches to teach them about planning.

A follow-up study was done where students from different universities were given a lecture on planning between assignments. Students were given programming problems in the pre-test and then assigned a new set of problems in the post-test. Students were also asked to produce two solutions for the post-test that embody different plans and preference-rank their solutions to see what criteria they used to evaluate their plans. We observed some positive impact, but not as much as we wanted in one course, so studies are continuing.

At this stage, I am in the process of developing (a) follow-up studies, (b) material for planning instruction, and (c) methods for measuring student planning performance. I am also building the literature for my dissertation proposal. The findings from the studies to be conducted will be used to inform my proposal which I hope to present at the end of the coming academic year.

7. EXPECTED CONTRIBUTIONS

The contributions of this work to computing education include a pedagogical framework for integrating programming instruction with planning instruction and methods of measuring student performance in planning, planning assessment, and transfer of learning in planning. Findings from studies conducted provides insights into students' planning and programming methodologies and their understanding of their own knowledge.

8. REFERENCES

[1] F. E. V. Castro and K. Fisler. On the Interplay Between Bottom-Up and Datatype-Driven Program Design. In *Proceedings of SIGCSE*, SIGCSE '16, pages 205–210, New York, NY, USA, 2016. ACM.

[2] M. de Raadt, M. Toleman, and R. Watson. Training Strategic Problem Solvers. *SIGCSE Bull.*, 36(2):48–51, June 2004.

[3] M. de Raadt, R. Watson, and M. Toleman. Teaching and Assessing Programming Strategies Explicitly. ACE '09, Darlinghurst, Australia, Australia, 2009.

[4] K. Fisler. The Recurring Rainfall Problem. ICER '14, pages 35–42, New York, NY, USA, 2014. ACM.

[5] O. Muller, D. Ginat, and B. Haberman. Pattern-oriented Instruction and Its Influence on Problem Decomposition and Solution Construction. ITiCSE '07, pages 151–155, New York, NY, USA, 2007. ACM.

[6] O. Seppälä, P. Ihantola, E. Isohanni, J. Sorva, and A. Vihavainen. Do We Know How Difficult the Rainfall Problem is? Koli Calling '15, pages 87–96. ACM, 2015.

[7] E. Soloway. Learning to Program = Learning to Construct Mechanisms and Explanations. *Commun. ACM*, 29(9):850–858, Sept. 1986.