# Fostering Strategic Knowledge and Program Comprehension Skills in Students Struggling with CS1

Violetta Lonati
lonati@di.unimi.it
Università degli Studi di Milano
Lab. CINI "Informatica e Scuola", Italy

Anna Morpurgo
morpurgo@di.unimi.it
Università degli Studi di Milano
Lab. CINI "Informatica e Scuola", Italy

## ABSTRACT

In order to support students struggling with learning to program, we designed new learning material, around two basic principles from recent research on introductory programming education: propose program comprehension tasks [4] besides writing tasks, and teaching strategic programming knowledge explicitly [5].

Such material may be of interest for high school teachers and academics teaching programming to novices. It was prepared and used in the context of a peer tutoring initiative targeted at undergraduate students of our CS degree program. We look for feedback and possible collaborators from other institutions, in order to carry out a methodologically sound investigation to measure the quality and effectiveness of the instructional material.

## CCS CONCEPTS

• **Social and professional topics → CS1**; **CS1**.

## KEYWORDS

CS1; program comprehension; strategic knowledge; goals and plans

## 1 MOTIVATION AND RELATED WORK

A significant feature that differentiates expert and novice programmers is their *strategic* knowledge [5], i.e., knowing stereotypical solutions to typical problems (so-called *plans* in Soloway's approach [6]) and being able to apply, tailor, and combine them to solve new problems. Many authors suggest explicitly teaching strategic knowledge [2, 6], since for many students this represents a real challenge.

Program comprehension, *i.e.,* the process of building a "mental model of a program" [4], is also an important skill to acquire in order to learn to program [1]. However, programming courses often neglect it, mostly focusing on code writing.

## 2 THE MATERIAL

The material we designed focuses on these topics: issues concerning input and plans for handling them; concepts and terminology related to goals and plans [6], in particular basic algorithmic iteration plans, as those listed in [2, 3]; the different ways in which plans can be composed to achieve aggregate goals.

ACM ISBN 978-1-4503-8062-1/21/03.
https://doi.org/10.1145/3408877.3439632

For each topic, a brief and schematic written presentation is followed by a sequence of decreasingly scaffolded exercises: students are assigned program comprehension tasks that help them identify and recognize plans and/or their composition in code snippets, while associating them with the corresponding goals; they are guided in completing, fixing or writing given plans; they are guided in analyzing program requirements in order to identify the goals and to design the appropriate plan/composition of plans; finally they are required to write programs from scratch, starting from requirements specifications written in natural language.

The material is available at https://aladdin.unimi.it/procosk.

## 3 VALIDATION

The general reception of the material was positive; students showed to have become aware of the importance of all the different programming skills, including the ability to reason about code, and they claimed that they did improve (see mean values in the table below). The goal/plan approach was very diversely received: "reflecting on programming abstractions" and "activities on recognizing patterns" were mentioned among the positive aspects; whereas someone critiqued the "focus on formalities (as recognizing patterns)".

| Self-evaluate your skills (from 1 = none, to 4 = total) | Before | After | Δ |
|---|---|---|---|
| Trace a piece of code on a given input | 1.94 | 2.88 | 0.94 |
| Detect errors in a piece of code | 2.06 | 3.00 | 0.94 |
| Apply to a new problem solution already seen | 1.94 | 3.00 | 1.06 |
| Recognize problems that are similar | 2.00 | 2.97 | 0.97 |
| Write programs starting from requirements spec. | 1.91 | 2.81 | 0.91 |

We will assess the effectiveness of the material and the feasibility of its use in classes. The study may include interventions based on the material, and assessed with pre- and post- tests or with the analysis of programs written by participants. Feedback on the study design and possibile collaborators are welcome.

## REFERENCES

[1] Tony Clear, J.L. Whalley, Phil Robbins, Anne Philpott, Anna Eckerdal, and M. Laakso. 2011. Report on the final BRACElet workshop: Auckland University of Technology, September 2010. *Journal of Applied Computing and Information Technology* 15, 1 (2011), 10.
[2] Michael de Raadt, Richard Watson, and Mark Toleman. 2009. Teaching and Assessing Programming Strategies Explicitly. In *Proceeding of ACE '09*. Australian Computer Society, Inc., AUS, 45–54.
[3] David Ginat, Eti Menashe, and Amal Taya. 2013. Novice difficulties with interleaved pattern composition. In *Prooceedings of ISSEP '13*, Vol. 7780 LNCS. Springer, 57–67.
[4] C. Izu, C. Schulte, A. Aggarwal, Q. Cutts, R. Duran, M. Gutica, B. Heinemann, E. Kraemer, V. Lonati, and C. Mirolo. 2019. Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories. In *ITiCSE-WGR '19*. ACM, New York, NY, USA, 27–52.
[5] Tanya J. McGill and Simone E. Volet. 1997. A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education* 29, 3 (1997), 276–297.
[6] Elliot Soloway. 1986. Learning to Program = Learning to Construct Mechanisms and Explanations. *Communication of the ACM* 29, 9 (1986), 850–858.