Modeling Programming Problem Solving Through Interactive Worked Examples

Dastyni Loksa
The Information School
University of Washington
Seattle, WA
dloksa@uw.edu

Andrew J. Ko
The Information School
University of Washington
Seattle, WA
ajko@uw.edu

Abstract

Problem solving is a critical programming skill, but most learning opportunities do not include instruction on it. Part of the reason for this may be due to the difficulty in modeling the cognitive and iterative nature of programming. In this paper we present the Problem Solving Tutor, a web based tool for delivering interactive worked examples that model the iterative and cognitive processes of programming.

Keywords Worked Example; Problem Solving; Programming

1 Introduction

Problem solving is a critical skill for novice programmers to develop, and involves complex self-regulation and metacognitive skills. And yet, these skills are rarely explicitly taught [5]. In many learning settings, novice programmers encounter problem solving process only indirectly through code examples or programming demonstrations, such as a teacher writing code in front of a classroom.

Domains like math and physics use worked examples to teach problem solving process, however, current programming worked examples are missing key elements that make them effective in those domains. In most domains worked examples include a problem statement, all steps necessary to craft a solution, and the final solution. Because programming is a highly iterative task requiring a lot of mental work it is challenging to represent all steps to the solution. Programming worked examples often only present a problem statement and final code with some line-by-line explanations of execution, or sub-goal labels [3]. What programming work examples lack are the cognitive and iterative steps toward the solution that allow learners to view and learn problem solving behaviors.

PLATEAU'17 Workshop on Evaluation and Usability of Programming Languages and Tools, October 23, 2017, Vancouver, CA

In this paper we present the Problem Solving Tutor, a tool to deliver programming worked examples that model cognitive and iterative problem solving behaviors.

2 Problem Solving Tutor

To address the many challenges of delivering process-based worked examples, we designed the Problem Solving Tutor (PST), depicted in Figure 1. The PST focuses on providing users with an interactive way to learn programming problem solving process. It follows a worked example model of teaching where an example programming problem is solved by an expert who demonstrates each step of the process while also providing rationale. The PST examples differ from current programming worked examples in two important ways: 1) it demonstrates each step an expert takes and 2) models the expert's decision making and reflection.

The PST provides an animated graphical representation of an expert programmer (Figure 1.1) as one of it's focal points. The expert is an important aspect of the PST because a learner's relationship with their teacher is a powerful mediator of the learning experience [4]. To make the expert engaging and person-like [1], the expert's facial expressions change to represent its current emotional state which ranges from disappointment to excitement.

The expert's speech bubble (Figure 1.2) is where all of the expert's dialogue is placed allowing the expert to communicate with the learner. Within the speech bubble the expert relates their thinking and rationale for their process and decisions. The expert demonstrates their planning, evaluation and reflection within the speech bubble simulating the expert using a think-aloud protocol. This models the expert's internal thought process and describes their mental work to the learner while solving the problem.

The PST also provides a visual representation of the expert's current state in a list of problem solving behaviors drawn from prior work [2] (Figure 1.3). The current behavior is always highlighted, animating to the next behavior whenever the expert reaches a step that has them enacting a different behavior. Showing these behaviors and transitions allows users to identify, at a glance, which of the problem solving behaviors the expert is currently engaging in, provides a visual reminder of what behaviors might be coming next, and which others behaviors are possible.

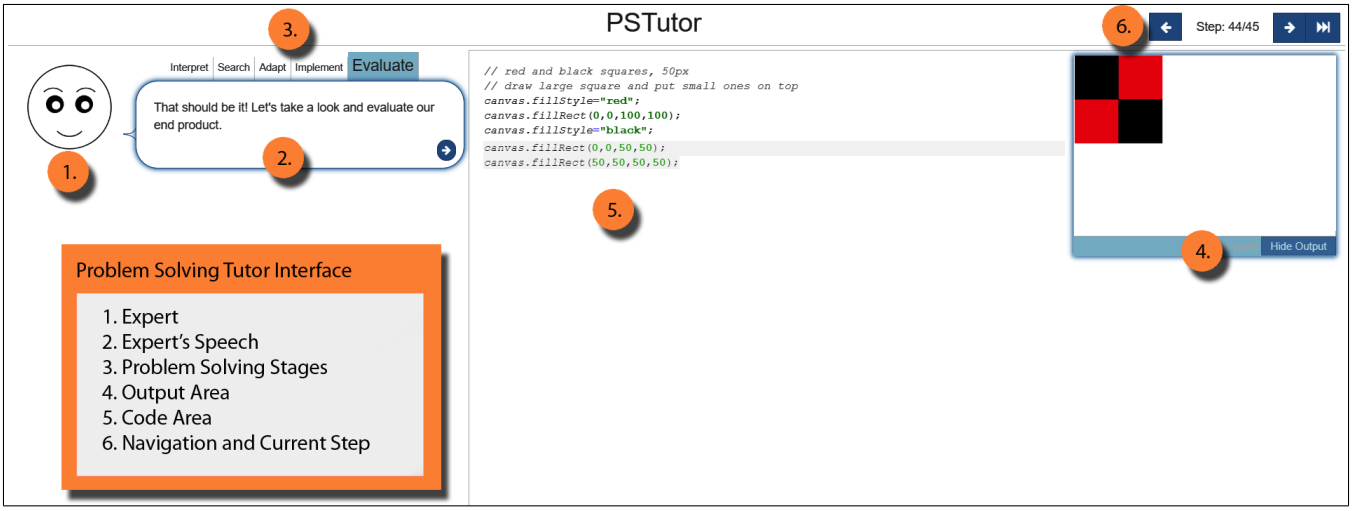

Figure 1. The Problem Solving Tutor interface. 1) The expert, 2) speech bubble, 3) problem solving behaviors, 4) output area, 5) code area, 6) navigation and current step.

The expert can prompt learners to reflect on the process using assessments. In assessments, the expert shifts the focus of from the code to the problem solving activities and poses a question to the learner such as, "Let's see what you know about the problem solving process. What do you think we should do next?" These multiple choice questions appear in the speech bubble where learners select an answer before receiving feedback about their selection, right or wrong. Learners can re-select answers and receive the feedback any number of times before progressing.

Figure 1.4 indicates the code area where the current state of the code is displayed. While the code is not editable by users, it includes animated "typing" of code by the expert during implementation steps allowing users to watch the expert write each line of code. Animations are skippable by proceeding to the next step in the example or by pressing the spacebar. The most recent block of code is also highlighted for easy identification, even after proceeding to the next problem solving steps.

To support code comprehension and evaluation, the PST includes a live output window (Figure 1.4) which shows the output for the code currently in the code area. In Figure 1 the output is showing the rendered JavaScript code from the code area (Figure 1.5). The output area can be collapsed to provide more visual space in the code area, however, when an example step requires the output for evaluation it will be automatically expanded into view.

Figure 1.6 shows the navigation bar. Learners can use their mouse or key bindings to navigate forward and backward through the example's steps or directly to the final solution if the example had been previously completed.

3 Conclusion

An inability to fully observe a programmer's problem solving process contributes to the difficulty many face when learning how to program. The Problem Solving Tutor presents programming worked examples in a way that allows learners to view each step of an expert's problem solving process. By modeling the process and including think-aloud rational and iteration on the code, learners are able to attempt to emulate the expert's process on future problems and with practice, learn to master them.

References

[1] Amy L. Baylor and Jeeheon Ryu. 2003. The effects of image and animation in enhancing pedagogical agent persona. *Journal of Educational Computing Research* 28, 4 (2003), 373–394.

[2] Dastyni Loksa, Andrew J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 1449–1461.

[3] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. 2015. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, 21–29.

[4] Erin E. O'Connor, Eric Dearing, and Brian A. Collins. 2011. Teacher-child relationship and behavior problem trajectories in elementary school. *American Educational Research Journal* 48, 1 (2011), 120–162.

[5] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer Science Education* 13, 2 (2003), 137–172.