Experiments with a Socratic Intelligent Tutoring System for Source Code Understanding

Zeyad Alshaikh, Lasang Tamang, Vasile Rus
University of Memphis
Memphis. TN 38152
{zlshaikh, ljtamang, vrus}@memphis.edu

Abstract

Computer Science (CS) education is critical in today's world, and introductory programming courses are considered extremely difficult and frustrating, often considered a major stumbling block for students willing to pursue computer programming related careers. In this paper, we describe the design of Socratic Tutor, an Intelligent Tutoring System that can help novice programmers to better understand programming concepts. The system was inspired by the Socratic method of teaching in which the main goal is to ask a set of guiding questions about key concepts and major steps or segments of complete code examples. To evaluate the Socratic Tutor, we conducted a pilot study with 34 computer science students and the results are promising in terms of learning gains.

Introduction

Computer Science (CS) education is critical in today's world where computing skills such as computer programming are an integral part of many disciplines including science, math, engineering, technology, and humanities such as political science.

Introductory programming courses are considered extremely difficult for most students (Lane and VanLehn 2004) and frustrating (Johnson 1990), and often considered a major stumbling block (Proulx 2000) for those willing to pursue programming-related careers. As a result, college CS programs suffer from high attrition rates (30-40%, or even higher) in introductory CS courses (Beaubouef and Mason 2005).

Research has shown that students in introductory programming courses experience many difficulties and often cannot write a fully functional program without help (Keim, Fulkerson, and Biermann 1997; Lane and VanLehn 2003). The root cause of all these difficulties is the inherent complexity of CS concepts, tasks, and the computing environment (Morrison, Margulieux, and Guzdial 2015). Furthermore, programming cannot be efficiently learned without considerable practice(Fenichel, Weizenbaum, and Yochelson 1970). However, a major problem is the fact that students cannot judge their code when it differs from a model answer. Therefore, students often need continued advice and feedback from experts (Hattori and Ishii 1999).

To overcome these challenges, we developed a dialogue based intelligent tutoring system (ITS) called Socratic Tutor (S.T.). S.T. is inspired by the Socratic instructional strategy which consist of a set of guiding questions meant to provide students a form of scaffolding targeting key aspects of a given instructional task. Furthermore, the S.T. relies on self-explanation theories of learning (Chi et al. 1994) by implementing instructional strategies such as eliciting self-explanations through Socratic questioning.

Our working hypothesis is that the S.T. positively impacts learning, self-efficacy, retention, and helps novice programmers achieve deeper understanding of programming concepts.

The rest of the paper is structured as in the following. First, we discuss background and related work and how the S.T. stands out among other programming ITS. Next, we discuss the architecture and design of S.T. in details. Finally, we illustrate the result of our pilot study in terms of learning gains and other metrics.

Background and Related Work

Socrates, a Greek philosopher, used a series of questions to guide students in their process of examining a target topic or concept. Socrates believed that these guiding dialogues would help his disciples, i.e., the learners, better understand a given topic, identify the incomplete understanding of key concepts, clarify any misunderstandings and correct misconceptions, therefore developing a deeper understanding of the target topic through some sort of guided self-discovery process with minimal help from the instructor/tutor. The Socratic method relies on a so-called direct line of reasoning (Chang et al. 2003) that emphasizes directing students' attention to key parts of a learning task thus triggering reasoning and explanation processes in students' minds which have been proven to be extremely beneficial in deep understanding tasks such as the ones we use in our case, i.e., understanding programming examples. Consequently, the Socratic tutoring method has been adapted by human tutors as well as computer tutors as early as 1977 (see Stevens and Collins' WHY system (Stevens and Collins 1977)).

Exploring the literature, one can find many ITSs for computer programming such as FIT Java Tutor(Gross and Pinkwart 2015), CIMEL ITS (Blank et al. 2005) and JITS (Sykes and Franek 2003). However, we focus on dialogue-based ITSs of which we mention the following previously developed systemss: Duke Programming Tutor (Keim, Fulkerson, and Biermann 1997), ProPL (Lane and VanLehn 2003), GENIUS (McCalla and Murtagh 1985) and PEA (Moore and Moore 1995). In the next paragraphs, we discuss each system and indicate how our S.T. system stands out relative to these prior efforts.

Duke Programming Tutor (DPT) is a multi-modal dialogue system that guides students through a standard programming lab to help them construct simple programs. The system uses a semantic network with a feature vector to model the domain and uses a temperature function to steer the dialogue. The DPT takes students' input and uses a simplistic minimum edit distance between their code and the goal program. Therefore, DPT is a task-oriented dialogue that leads students to achieve a set of goals, and as a result, write a simple Pascal program. In our case, we do not focus on constructing programs but rather on understanding computer programs.

The second ITS example is PROgram PLanner (PROPL) which focuses on programming design and problem solving. PROPL uses Coached Program Planning (CPP) (Lane and VanLehn 2003) as its tutoring strategy which consists of eliciting problem decomposition from students. Therefore, the tutor gives students a problem and asks them to identify the following elements: (1) goal(s), which are the objectives declared by the main program, (2) schema, which is the method to accomplish the goal(s), and (3) objects, which are the data that is required by the program. The final product from the interaction is a pseudocode that can be translated into a program.

The goal of both DPT and PROPL is to help students solve programming tasks and generate the corresponding code (pseudo or otherwise), that is, the emphasis is on planning and designing phases. Furthermore, these systems are task-oriented and are focused on guiding students toward final goal.

The two other example systems, GENIUS and PEA, can be viewed as helping systems. GENIUS interacts with the students in order to help them fix syntactic errors. The interaction involves keeping the student engaged in a dialogue, guiding them to identify the errors on their own. The interaction is limited to yes/no questions and "I don't know" responses. Finally, the PEA system helps students improve their coding style by providing advice on how to make a program more readable and maintainable. The PEA system is able to answer questions the student may have regarding suggested changes in code. It should be noted that in our case we do not provide code examples that have errors as this could be potentially frustrating for novices, i.e., students in CS1 and CS2. We do plan to add debugging tasks in the future but for that we need a more fine-grained learner model that will help us decide for which students such debugging tasks are appropriate.

Unlike these other ITSs, our S.T. is able to engage students in an active dialogue, automatically assess their responses and thereofore knowledge and also clarify any misconceptions. These are major features based on which the S.T. computer tutor is different the other ITSs targeting computer programming related learning tasks.

Socratic Tutor for Programming

As already pointed out earlier, S.T. is a web-based dialogue ITS that can help novice programmers better understand programming concepts. S.T is programming language independent and can be used to teach any programming language.

The architecture of the S.T. consists of a student model, a domain model, an interaction model, and a dialogue and natural language understanding (NLU) engine (Banjade et al. 2015). The student model contains student's level of mastery with respect to the target topic and the domain model consists of a list of topics to be covered and in what order. The interaction model is defined by the dialogue policy which selects at each moment in the dialogue what the next S.T. dialogue move should be. The policy is modulated by the output of the NLU engine which is basically a similarity engine comparing the student response to an ideal response. The dialogue policy can be customized for each tutoring session.

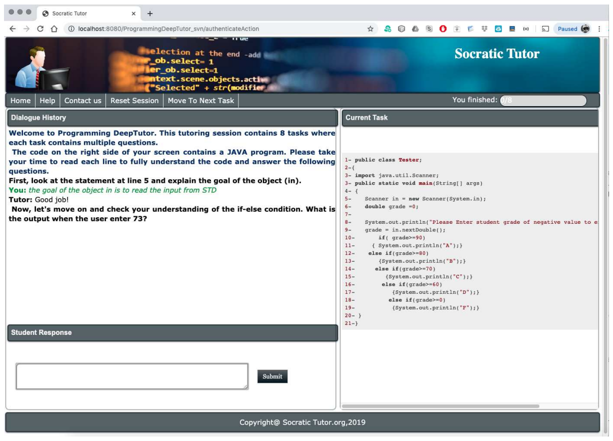
The interface of the S.T consists of three main area: (1) a code area, where the source code example is shown, (2) a dialogue area, where the dialogue history is shown, and (3) a student response area which is where students type their input which could be greetings, answers to tutor's questions or their ask their own questions.

The main type of instructional tasks is source code understanding and output prediction. Therefore, the S.T shows a code example and asks the learner to explain the code and predict its output. Then, the S.T guides the student through dialogue, e.g., by asking questions, to focus on the major programming concepts and steps in each code example. The set of questions were manually designed by experts following the Socratic method's guidelines. Furthermore, a major goal of the S.T. system is to evaluate the students' knowledge and detect any misconceptions as mentioned previously.

The S.T system provides support to learners in the form of hints in two major instances: (1) when a student asks explicitly for help or (2) when the answer is incomplete or incorrect. In both cases, the S.T starts a 3-level feedback loop.

At the first level, the tutor briefly explains the programming concept and asks the student to try to answer the original question. If the student fails, the tutor provides a hint in the form of a fill in the blank question. This type of hint limits the student's answers and draws more attention toward the key part of the solution. If both levels 1 and 2 hints fail to elicit a correct response from the learner, then the S.T. tutor provides level-3 feedback in the form of a multiple-choice question. Finally, if the student cannot give the correct answer, the system will present the solution and move to the next question.

It should be noted that if the learner provides a correct answer, then the system gives positive feedback such as "Well



Welcome to Programming DeepTutor. This tutoring session contains 8 tasks where each task contains multiple questions.
The code on the right side of your screen contains a JAVA program. Please take your time to read each line to fully understand the code and answer the following questions.
First, look at the statement at line 5 and explain the goal of the object (in).
You: *the goal of the object in is to read the input from STD*
Tutor: Good job!
Now, let's move on and check your understanding of the if-else condition. What is the output when the user enter 73?

```
1- public class Tester;
2-{
3- import java.util.Scanner;
3- public static void main(String[] args)
4- {
5-    Scanner in = new Scanner(System.in);
6-    double grade =0;
7-
8-    System.out.println("Please Enter student grade of negative value to e
9-    grade = in.nextDouble();
10-       if( grade>=90)
11-     { System.out.println("A");}
12-    else if(grade>=80)
13-       {System.out.println("B");}
14-      else if(grade>=70)
15-         {System.out.println("C");}
16-        else if(grade>=60)
17-           {System.out.println("D");}
18-          else if(grade>=0)
19-           {System.out.println("F");}
20- }
21-}
```

Figure 1: Socratic Tutor interface

done!" and "Great Job!" to promote engagement and confirm to the learner that they answered correctly the corresponding question.

Experimental Setup

We evaluated the effectiveness of the S.T by conducting a pilot study with 34 students attending introductory courses in Computer Science at an urban university in southeast Asia. The study tried to answer following research questions.

- RQ1: How much do students learn if using S.T.?
- RQ2: Do students with lower prior knowledge learn more?
- RQ3: How does self-efficacy affect students' learning processes?

Materials

Participants were asked to fill a background questionnaire and a self-efficacy survey at the beginning of the experiment. The self-efficacy survey contained 11 questions related to each programming concept subjects will encounter during the tutoring session. Then, the participants were assigned a pre- and post-test. The pre and post-test have similar level of difficulty and contains 9 JAVA programs. The participants have to predict the output of each program.

Procedure

The experiment was conducted in a computer lab under the supervision of experimenters. First, participants were debriefed about the purpose of the experiment and asked to read and sign a consent form if they agreed to proceed. Then, they took a background questionnaire, self-efficacy survey, and a pre-test. Once they have finished all these initial assessments, an approximately 60-minute tutoring session with the S.T. started in which they worked on 9 Java code examples. Finally, they took a post-test which has a similar format and difficulty level as the pre-test.

Table 1: Pre and post-test scores: mean, standard deviation, improvement, and learning gain

		pre-test		post-test			
Section	n	Mean	SD	Mean	SD	Improvement	Learning gain
All	34	75.82	11.6	88.4	9.71	12.58	52.03
TOP	17	83.99	7.58	91.83	7.89	7.84	48.97
BOTTOM	17	67.65	8.84	84.97	10.35	17.32	53,54

* Improvement = post-test - pertest.

Table 2: Self-efficacy: mean, standard deviation, improvement, and learning gain

		Self-efficacy			
Section	n	Mean	SD	Improvement	Learning
TOP	17	80.77	11.72	15.81	53.85
BOTTOM	17	42.38	11.08	14.10	51.01

Assessment

Each question in the pre and post-test were scored with 1 when the answer provided by the student was correct and 0 otherwise. Based on this rubric and the student responses, we computed a learning gain (LG) score as follows (Marx and Cummings 2007).

- If post-test >pre-test, gain = (post-test - pre-test)/(100 - pre-test)
- If post-test <pre-test, gain = (post-test - pre-test)/pre-test
- If post-test=pre-test= 0 or 100, drop the cases
- If post-test=pre-test, gain = 0

Results

Learning Gains To understand the overall effectiveness of S.T, we report a improvement metric and LGs. Table 1 shows that participants had an average pre-test score of 75.82% and an average post-test score of 88.4%. The average improvement is 12.58% ($p < 0.01$) and the LG score is 52.03%. Thus, the overall increase in knowledge is promising, validating the effectiveness of the implemented proposed Socratic strategy.

To answer the second research question, **RQ2**, we divided the participants into two groups (TOP vs. BOTTOM) based on the mean pre-test score. The TOP group had an average pre-test score of 83.99% and average post-test score of 91.83% resulting in a 7.84% improvement and 48.97% LG. On the other hand, the BOTTOM group had an average pre-test score of 67.65% and an average post-test score of 84.97% resulting in 17.32% improvement and 53% LG. Therefore, participants with lower prior knowledge learn more with a difference of 9.94% in improvement and 4.57% in LG ($p <0.01$).

Self-efficacy: To answer RQ3 we divided the participants again into TOP and BOTTOM groups based on the mean score of the self-efficacy survey responses. The results in table 2 show participants with higher self-efficacy scores have an increase of 1.71% in improvement and 2.84% in LG. However, the p-value ($p > 0.05$) but the increase is not statistically significant.

Table 3: 3-level Feedback

Level	Type	Helpful	Not helpful
1	Concept explanation	62.02%	37.98%
2	Fill in the Blank	76.47%	23.53%
3	Multiple choice	69.05%	30.95%

*Helpful means the student correctly answer the question after receiving the feedback

Feedback: To evaluate the effectiveness of our 3-level feedback approach, we analyze students responses for each level. Table 3 shows that 62.02% of the students who received level-1 feedback were able to give the correct answer. The percentage increased by 14.45% in level-2 feedback and by 7.03% in level-3 feedback.

Conclusions and Future Work

Promoting deep understanding of programming concepts is the primary focus of our Socratic ITS. The S.T. tutor is a dialogue based ITS that incorporates a Socratic line of questioning to steer and guide learner's attention to the important parts of a target code example. The S.T system is programming language independent, therefore can be used to tutor any programming language.

S.T is part of our larger agenda of investigating the role of Socratic instructional strategy for source code comprehension and computer programming learning. We are planning to carry out a controlled experiment to compare the Socratic strategy with other known strategies.

Acknowledgments

This work as partially funded by the National Science Foundation under the award #1822816 (Collaborative Research: CSEdPad: Investigating and Scaffolding Students' Mental Models during Computer Programming Tasks to Improve Learning, Engagement, and Retention) to Dr. Vasile Rus. All opinions stated or implied are solely the authors' and do not reflect the opinions of the funding agency.

References

Banjade, R.; Niraula, N. B.; Maharjan, N.; Rus, V.; Stefanescu, D.; Lintean, M.; and Gautam, D. 2015. Nerosim: A system for measuring and interpreting semantic textual similarity. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, 164–171.

Beaubouef, T., and Mason, J. 2005. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin* 37(2):103–106.

Blank, G.; Parvez, S.; Wei, F.; and Moritz, S. 2005. A web-based its for oo design. In *Proceedings of Workshop on Adaptive Systems for Web-based Education at 12th International Conference on Artificial Intelligence in Education (AIED'2005). Amsterdam, the Netherland*, 59–64. Citeseer.

Chang, K.-E.; Sung, Y.-T.; Wang, K.-Y.; and Dai, C.-Y. 2003. Web/spl i. bar/soc: a socratic-dialectic-based collaborative tutoring system on the world wide web. *IEEE Transactions on Education* 46(1):69–78.

Chi, M. T.; De Leeuw, N.; Chiu, M.-H.; and LaVancher, C. 1994. Eliciting self-explanations improves understanding. *Cognitive science* 18(3):439–477.

Fenichel, R. R.; Weizenbaum, J.; and Yochelson, J. C. 1970. A program to teach programming. *Communications of the ACM* 13(3):141–146.

Gross, S., and Pinkwart, N. 2015. Towards an integrative learning environment for java programming. In *2015 IEEE 15th International Conference on Advanced Learning Technologies*, 24–28. IEEE.

Hattori, N., and Ishii, N. 1999. An extended educational system for programming and its evaluation. *Information and Software Technology* 41(8):525–532.

Johnson, W. L. 1990. Understanding and debugging novice programs. *Artificial intelligence* 42(1):51–97.

Keim, G.; Fulkerson, M.; and Biermann, A. 1997. Initiative in tutorial dialogue systems. In *Proceedings of the American Association for Artificial Intelligence (AAAI) Spring Symposium on Computational Models for Mixed-Initiative Interaction*.

Lane, H. C., and VanLehn, K. 2003. Coached program planning: dialogue-based support for novice program design. In *ACM SIGCSE Bulletin*, volume 35(1), 148–152. ACM.

Lane, H. C., and VanLehn, K. 2004. A dialogue-based tutoring system for beginning programming. In *FLAIRS Conference*, 449–454.

Marx, J. D., and Cummings, K. 2007. Normalized change. *American Journal of Physics* 75(1):87–91.

McCalla, G., and Murtagh, K. 1985. *GENIUS: An experiment in ignorance-based automated program advising*. University of Saskatchewan, Department of Computational Science.

Moore, J. D., and Moore, J. D. 1995. *Participating in explanatory dialogues: interpreting and responding to questions in context*. MIT press Cambridge, MA.

Morrison, B. B.; Margulieux, L. E.; and 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*, 21–29. ACM.

Proulx, V. K. 2000. Programming patterns and design patterns in the introductory computer science course. In *ACM SIGCSE Bulletin*, volume 32(1), 80–84. ACM.

Stevens, A. L., and Collins, A. 1977. The goal structure of a socratic tutor. In *Proceedings of the 1977 annual conference*, 256–263. ACM.

Sykes, E. R., and Franek, F. 2003. A prototype for an intelligent tutoring system for students learning to program in java (tm). In *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education*, 78–83.