Paper ID #26921

PRIME: Engaging STEM Undergraduates in Computer Science with Intelligent Tutoring Systems

Dr. James C. Lester, North Carolina State University

James C. Lester is a Distinguished Professor of Computer Science and Director of the Center for Educational Informatics at North Carolina State University. He is a Fellow of the Association for the Advancement of Artificial Intelligence (AAAI). His research on personalized learning technologies ranges from intelligent tutoring systems and game-based learning environments to affective computing, computational models of narrative, and natural language tutorial dialogue. The adaptive learning environments he and his colleagues develop have been used by thousands of students in K-12 and college classrooms throughout the US and internationally.

Kristy Elizabeth Boyer, University of Florida
Dr. Eric N. Wiebe, North Carolina State University

Dr. Wiebe is a Professor in the Department of STEM Education at NC State University and Senior Research Fellow at the Friday Institute for Educational Innovation. Dr. Wiebe works on many different facets of STEM Education, including the design and evaluation of innovative uses of computing technologies in STEM instructional settings, the use of multimedia tools for teaching and learning, and student engagement and persistence in STEM career pathways. Specific research programs include: the use of game-based environments to learn about computational thinking, the use of intelligent agents to support science learning in classrooms, and basic research in the how instructional technologies shape student engagement and learning. Since the integration of these technology tools are essential for their effective use, research is also being pursued at large scales, looking at how specific technologies influence teaching and learning at the classroom and school level and how schools and teachers can be supported to change practice in order to maximize the potential of these new technologies. Similarly, Dr. Wiebe is interested in how these innovative tools can be used in and outside of classrooms to enhance student interest in STEM learning and career opportunities.

Dr. Bradford Mott, North Carolina State University

Bradford W. Mott received his B.S., M.C.S., and Ph.D. in Computer Science from North Carolina State University, where he is currently a Senior Research Scientist in the Center for Educational Informatics. His research focuses on game-based learning environments, intelligent tutoring systems, computer science education, and computational models of interactive narrative.

Mr. Andy Smith, North Carolina State University

Andy Smith is a Research Scientist with the Center for Educational Informatics at North Carolina State University. His research focuses on the development and analysis of educational technology.

PRIME: Engaging STEM Undergraduates in Computer Science with Intelligent Tutoring Systems

Introduction

This NSF IUSE project focuses on the design, development, and evaluation of PRIME, an intelligent tutoring system for introductory computing. We define *computing* as the creative design, implementation, and analysis of artifacts to solve computational problems. Leveraging advanced intelligent tutoring systems technologies, PRIME will provide integrated problem-solving and motivational support dynamically tailored to individual students over the course of their problem-solving sessions. PRIME is being designed to address the specific needs of STEM undergraduates in introductory computing courses. These students, most of whom are not computer science majors, exhibit a wide range of initial capabilities and dispositions toward computing. Many have had limited previous experience with computing, a problem that is particularly acute for women and underrepresented minorities. PRIME is being designed to address these important individual differences.

The project has the overarching objective of *transforming introductory computing for STEM majors by creating an intelligent tutoring system that provides individualized problem-solving and motivational support in order to improve the learning experience for these students.* PRIME will track each student's progress in learning computer science concepts and techniques while providing real-time feedback, multiple levels of hints, and customized problem-solving advice throughout students' learning interactions with the system. It is being evaluated in introductory computing courses at North Carolina State University and Florida Agricultural and Mechanical University, a Historically Black University. PRIME evaluations will center on investigating its effect on student learning of computer science (*analyzing problems*, *creating models and abstractions, and building and refining programs),* and its effect on students' attitudes towards computer science (*self-efficacy for computing* and *interest in computing*).

Research Context

Creating effective introductory computing courses is widely recognized as a central challenge for computer science education. These courses, particularly those that must serve a diverse population of students outside of computer science, are difficult to design and present tremendous challenges for the students who take them. For example, many novices hold mental models that are not compatible with learning to program [1]. These challenges have led to a broad range of instructional innovation, many of which are focused on problem solving as a central mechanism for developing computer science knowledge and expertise. Flipped classrooms, in which problem solving comprises the majority of classroom time [2], [3], and a closely related approach, lab-centric instruction [4], have shown promise and are under active investigation, as are problem-based learning [5] and game-based learning [6], [7]. Investigating how students come to understand the fundamental concepts of computing [8], how to support the learning of these concepts [9],and determining the most effective ordering of concepts has also been extensively studied (e.g., [10]). However, limited instructional resources and the lack of capacity to provide individualized support is a longstanding problem. The PRIME project is addressing this issue with an intelligent tutoring system and virtual tutors that provide individualized support.

The problem-solving activities within introductory computing pose significant challenges to novice students, who need substantial practice to develop proficiency. A variety of tools have been created to support problem solving, including online support for Java programming [11], for building and visualizing Python programs [12], for visualizing JavaScript runtime behavior [13], for real-time collaboration and feedback from instructors [14], and for leveraging a gaming context to foster engagement [15]. While some introductory computing courses utilize industry-standard integrated development environments (IDEs) such as Eclipse or Netbeans, others use IDEs designed for novices, such as BlueJ [16], and some try to bridge the gap between syntax-heavy languages and block-based languages [17]. Even the best IDEs, however, do not provide step-by-step support for problem solving. To provide this support, the PRIME project will build upon the broad range of personalized learning functionalities provided by intelligent tutoring systems, including the rich literature on intelligent tutoring systems that support learning to program (e.g., [18], [19]).

System Design

The design process for PRIME began by first developing a set of tasks to use as the basis for teaching programming concepts. To do so, we reviewed the syllabi for introductory programming courses from the fifty top-rated undergraduate computer science programs in the US [20]. From this review, we extracted the set of topics that are typically covered in the first five weeks of the courses and the order in which they are covered: 1) Input/Output, Variables, and Loops; 2) Functions, Parameters, and Return Values; 3) Conditional Execution; 4) String Manipulation and Basic Data Structures; and 5) Search and Sort Algorithms. The work presented in this paper focuses on Units 1-3, with the full list of topics enumerated in Table 1 below.

Table 1. CS Topics Covered in the Research Study

Unit	Topics
1	Environment tutorial, Input/Output, Numeric data types, Expressions (math), Variables, Iteration (definite)
2	Abstraction, Functions (methods), Parameters
3	Boolean data types, Conditionals, Iteration (indefinite), Debugging

Each unit (typically covered in a week) consists of multiple sequential activities. Units 1 and 2 of PRIME consist of 7 activities each, with Unit 3 consisting of 6 activities. Within each init, activities progressively build upon concepts and require students to build more complex programs to solve increasingly difficult tasks.

When selecting a block-based programming language for PRIME, we evaluated several alternatives for block-based programming platforms. We selected Google's Blockly framework [21] due to its ease of customization and its ability to translate block programs into text-based (e.g., Python) equivalents. The main user interface includes the *Blockly* panel, the *Console* panel, the *Feedback* panel, and the *Instructions* panel. The *Blockly* panel consists of a visual coding widget with the block-based coding workspace and toolbox of available blocks. The default workspace has been augmented with a "Start" block, which serves a similar purpose to the "Main" function or method in other programming languages. The toolbox varies for each task, gradually adding more blocks as the students complete tasks and are introduced to new topics. This approach is based on prior research which suggests introducing new blocks only as needed may reduce extraneous cognitive load [22] and increase interface usability for novice learners [23].

The *Console* panel contains a "Run" button and shows the output generated from running the program. An input window also appears in this panel if a program prompts the user for input. Finally, the *Instructions* panel contains step-by-step instructions for a given task. This type of instruction format is common for intelligent programming tutors [24], though rarely found in to block-based programming environments. In addition to navigation buttons, this panel also contains positive feedback and links to the next task when a student has successfully completed the current task. Task completeness is checked every time a student runs their program, and is based primarily on a set of exemplar cases authored by the research team for each activity. The *Feedback* panel contains the "Get Hint" button, allowing students to request a textual hint. Hints check various aspects of the student code, including the presence or absence of certain blocks, structural features such as whether code is connected to the Start block, as well as the content of parameters and fields of certain blocks. Multiple hints were authored for each activity, based on common errors identified from previous data collections and pilot testing.

Outcomes

Through PRIME 's development, we have designed and developed a block-based programming environment to introduce programming novices to computer science concepts. PRIME currently consists of 20 activities, split into three units, covering topics typical of an introductory programming course for non-CS majors. We have also successfully integrated PRIME into the Moodle and Canvas learning management systems, facilitating the deployment of PRIME into the classroom. As a result, over 500 introductory engineering students at North Carolina State University and over 100 web programming and engineering students at Florida A&M University (a historically Black university) have interacted with the PRIME learning activities during their coursework.

Our studies have also yielded important design recommendations. We have found that decomposing activities into subtasks benefits students, as investigating how these subtasks should be presented (freely available, or after completing subgoals). Initial findings show that adaptively enabling subtasks can lead to improved code quality in both that task, and future tasks within the same learning unit (in preparation). Recently, we have also investigated how students use of the adaptive hinting system and programming behaviors during their interactions with PRIME can be used to train predictive student models capable of identifying struggling students and providing additional adaptive scaffolding (in preparation).

As part of our data collection studies, we have also conducted focus groups and interviews with the students and instructors to understand how we can improve the system to meet their learning needs. Students have reported a desire for design features such as realistic exercises, on-demand hinting and support, and access to more challenging activities. Students have overall expressed positive impressions of PRIME, particularly liking the adaptive support and real-time feedback during coding activities. Students have also generally responded positively to the "real-world" themes of activities. However, students have also expressed some displeasure with aspects of the

PRIME system, such as the quantity and usefulness of the hints on certain exercises, as well as a perception that the system is forcing them toward one solution. We continuously take student feedback into consideration as we work on improving the PRIME application with each new version.

Conclusion

While mitigating some of the difficulty of introductory programming exercises, block-based programming environments could benefit considerably from leveraging lessons and techniques from other instructional fields such as intelligent tutoring systems. The PRIME system addresses these issues through the creation of a block-based programming environment focused on novice programmers. Over the past two years, we have worked with students and instructors to develop the PRIME platform and its learning activities. These activities are aligned with the early weeks of typical CS0 curriculums, and provide students with features and supports including incremental instructions and adaptive supports tailored to the state of the student's current program. Facilitated by integration with LMS systems such as Canvas and Moodle, PRIME has been used by over 600 students at NC State and Florida A&M University helping to refine the system for future iterations.

As PRIME moves forward we will look to both expand the current set of activities, as well as move forward in the development and integration of data-driven student models capable of driving adaptive scaffolding within the existing PRIME activities. Additionally, we will investigate alternative approaches to transition students to text-based coding activities, utilizing Blockly's ability to generate code from multiple languages from block-based programs. Finally, we are looking to integrate PRIME into more classrooms and develop more refined assessments to better evaluate PRIME'S effects on both student CS knowledge and attitudes.

References

[1] J. Ferguson, M. Roper, M. Wood, and L. Ma, “Investigating and improving the models of programming concepts held by novice programmers,” *Comput. Sci. Educ.*, vol. 21, no. 1, pp. 57–80, 2011.

[2] C. Latulipe, N. B. Long, and C. E. Seminario, “Structuring Flipped Classes with Lightweight Teams and Gamification,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE ’15*, 2015, pp. 392–397.

[3] K. Lockwood and R. Esselstein, “The inverted classroom and the CS curriculum,” in *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE ’13*, 2013, p. 113.

[4] N. Titterton, C. M. Lewis, and M. J. Clancy, “Experiences with lab-centric instruction,” *Comput. Sci. Educ.*, vol. 20, no. 2, pp. 79–102, 2010.

[5] S. B. Fee and A. M. Holland-Minkley, “Teaching computer science through problems, not solutions,” *Comput. Sci. Educ.*, vol. 20, no. 2, pp. 129–144, 2010.

[6] A. Iosup and D. Epema, “An experience report on using gamification in technical higher education,” in *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE ’14*, 2014, pp. 27–32.

[7] D. Toth and M. Kayler, “Integrating Role-Playing Games into Computer Science Courses as a Pedagogical Tool,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE ’15*, 2015, pp. 386–391.

[8] P. Hubwieser, J. Magenheim, A. Mühling, and A. Ruf, “Towards a conceptualization of pedagogical content knowledge for computer science,” in *Proceedings of the ninth annual international ACM conference on International computing education research - ICER ’13*, 2013, p. 1.

[9] L. J. Barker, M. O’Neill, and N. Kazim, “Framing classroom climate for student learning and retention in computer science,” in *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE ’14*, 2014, pp. 319–324.

[10] A. Ehlert and C. Schulte, *Empirical comparison of objects-first and objects-later*. 2009.

[11] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, “CodeWrite: Supporting student-driven practice of Java,” in *42nd ACM Technical Symposium on Computer Science Education, SIGCSE 2011*, 2011, pp. 471–476.

[12] P. J. Guo, “Online Python Tutor: EmbeddableWeb-Based Program Visualization for CS Education,” in *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE ’13*, 2013, p. 579.

[13] T. Lieber, J. R. Brandt, and R. C. Miller, “Addressing misconceptions about code with always-on programming visualizations,” in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI ’14*, 2014, pp. 2481–2490.

[14] J. Vandeventer and B. Barbour, “CodeWave: a real-time, collaborative IDE for enhanced learning in computer science,” *Proc. 43rd ACM Tech. Symp. Comput. Sci. Educ. - SIGCSE ’12*, pp. 75–80, 2012.

[15] S. Kurkovsky, “Mobile game development: Improving student engagement and motivation in introductory computing courses,” *Comput. Sci. Educ.*, vol. 23, no. 2, pp. 138–157, 2013.

[16] M. Kolling, B. Quig, A. Patterson, and J. Rosenberg, “The BlueJ System and its Pedagogy,” *Comput. Sci. Educ.*, vol. 13, no. 4, pp. 249–268, 2003.

[17] Y. Matsuzawa, T. Ohata, M. Sugiura, and S. Sakai, “Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE ’15*, 2015, pp. 185–190.

[18] A. T. Corbett, J. R. Anderson, and E. G. Patterson, “Student Modeling and Tutoring Flexibility in the Lisp Intelligent Tutoring System,” in *ITS ’90*, 1990, pp. 83–106.

[19] W. Jin and A. T. Corbett, “Effectiveness of Cognitive Apprenticeship Learning (CAL) and Cognitive Tutors (CT) for Problem Solving Using Fundamental Programming Concepts,” in *SIGCSE ’11*, 2011, pp. 305–310.

[20] Y. Matsuzawa, T. Ohata, M. Sugiura, and S. Sakai, “Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE ’15*, 2015, pp. 185–190.

[21] M. Stanger and E. Martin, “The 50 best computer-science and engineer- ing schools in America,” 2015. [Online]. Available: http://www.businessinsider.com/best-computer-science-engineering-schools-in-america-2015-7/.

[22] N. Fraser, “Google blockly-a visual programming editor,” *https://developers.google.com/blockly/. Accessed April (2016).*, 2013. .

[23] A. Renkl and R. K. Atkinson, “Structuring the transition from example study to problem solving in cognitive skill acquisition: A cognitive load perspective,” *Educ. Psychol.*, vol. 38, no. 1, pp. 15–22, 2003.

[24] F. J. Rodriguez, K. M. Price, J. Isaac, K. E. Boyer, and C. Gardner-Mccune, “How block categories affect learner satisfaction with a block-based programming interface,” in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2017, vol. 2017–Octob, pp. 201–205.

[25] T. Crow, A. Luxton-Reilly, and B. Wuensche, “Intelligent tutoring systems for programming education,” in *Proceedings of the 20th Australasian Computing Education Conference on - ACE ’18*, 2018, pp. 53–62.