Guest Editorial Special Issue on Computing in Engineering

WITH the increasing ubiquity of computing, more engineering programs now require their students to take one or more computing courses. At institutions where significant numbers of engineering students take computer courses, computing instructors and educators often assume roles that have them teaching their computing courses for non-majors as service courses, dealing with students with diverse backgrounds and varied motivations. Although teaching computing courses to non-majors is not new, the increasing importance of computing in future undergraduate engineering curricula poses new challenges. In this Special Issue, four of the papers offer insight into, and give evidence of, teaching computing to engineering students; the fifth paper investigates computational modeling abilities in conceptual understanding of electric circuits.

This collection of papers reports findings based on data collected using a variety of methods: focus groups, classroom observations, survey questions, interviews, pre- and post-tests, student GPA and more. These methods each provide a unique perspective on understanding how instructors or students view a particular treatment or course item, and on assessing student performance or achievement in particular topics. Each paper's discussion of results is based on a mixed subset of such methods. Supplementing the data collection methods, several papers also pursue statistical analyses to establish the significance of any impacts resulting from the proposed approaches or treatment. These methods and analytical approaches serve as exemplars for other researchers and practitioners to adopt and utilize in their own research studies.

Each paper in the collection has an extensive section on related work, prior research, and/or conceptual frameworks. The references cited in these organized summaries, and the future directions the authors indicate for further research on the findings or for steps to address limitations or issues encountered, both serve as valuable resources for readers and practitioners wishing to pursue and explore pedagogical and curricular innovations.

The first paper in this special issue, by Clark and Dickerson, reports on the use of active learning techniques such as think-pair-share, minute papers, and pair programming in digital logic and computer organization/assembly language, applied in two courses in an introductory computing sequence. This paper offers evidence that it can be straightforward and effective to administer or incorporate simple yet effective active learning techniques into classrooms to teach computing to engineering students, as instructor preparation in using such techniques can be done in a one-time workshop.

The second paper in this special issue, by Giacaman and De Ruvo, reports on the use of an integrated development environment plug-in called Active Classroom Programmer (ACP) that allows the instructor to switch between "teaching to" and "programming with" students. This facilitates collaborative programming between the instructor and students, in which timely interventions of real-time programming demonstrations help reinforce learning outcomes more concretely. This strengthens the constructive alignment between learning outcomes and their associated activities without imposing significant constraints on the instructor's lectures. Results in a CS2 and a graduate compiler course show students performed better with higher ACP-usage.

The third paper in this special issue, by Smith, Guigliano, and DeOrio, reports on the long-term effects of pair programming in subsequent courses. The study examines close to 2,500 students in an introductory CS sequence. The findings show that there is a positive relationship between pair programming in an introductory course (i.e., CS2) and higher project scores in a future course (i.e., CS3) where all students worked alone, with students with the lowest GPAs experiencing the greatest benefits.

The fourth paper, by Peteranetz, Flanigan, Shell, and Soh, reports on the use of computational creativity exercises (CCEs)—group-based learning activities that integrate computational thinking and creative thinking—to improve student learning and performance in introductory computer science (CS1) courses for engineering students. Findings from this paper provide additional evidence that these exercises have an independent effect on student learning in CS1. The authors expand on previous quasi-experimental findings, that the CCEs improve engineering students' learning in CS1, by demonstrating effects in a Propensity Score Matched trial, which allows for strong causal conclusions; these results offer greater confidence that the effect will generalize to other classrooms.

The fifth paper, by Ortega-Alvarez, Sanchez, and Magana, reports on the evaluation of the use of a multi-representation approach that includes both computational and mathematical representations in solving problems and conceptual understanding in electric circuits. Students were asked to apply their programming skills to create a computer model with the mathematical equations they produced. Correlation between the use of representation and conceptual understanding was then computed. Findings indicate that the number and quality of students' representations correlate positively with their conceptual understanding. In particular, the quality of the computational representations was found to be highly, and

Digital Object Identifier 10.1109/TE.2018.2828958

significantly, correlated with the correctness of students' answers to qualitative reasoning questions.

In closing, we hope that this Special Issue on Computing in Engineering will, first, encourage computing faculty to adopt similar approaches in teaching computing in engineering, investigate and explore more effective approaches, and together improve the practice of teaching computing in engineering, and second, suggest and inspire future research directions for computing education in engineering to advance this area's pedagogical knowledge.

LEEN-KIAT SOH, *Guest Editor*
Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE 68588 USA
(e-mail: lksoh@cse.unl.edu)

STEPHEN COOPER, *Guest Editor*
Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE 68588 USA
(e-mail: scooper@cse.unl.edu)

Leen-Kiat Soh (M'98) received the B.S. degree (with Highest Distinction) and the M.S. and Ph.D. degrees (Hons.) from the University of Kansas, Lawrence, KS, USA, all in electrical engineering. He is currently a Full Professor with the Department of Computer Science and Engineering, University of Nebraska, Lincoln, NE, USA, where he is also the Dean's Professor of Teaching and Learning. His current research interests include multiagent systems and intelligent agents, computer-aided education systems, computer science education, and image processing. He has authored over 180 peer-reviewed journal and conference publications in these areas. His current research focuses on using computational creativity to improve student learning and developing agent-powered or learning-enabled intelligent applications. He has been serving as an Associate Editor for the IEEE TRANSACTIONS ON EDUCATION since 2014. He is a member of the ACM and AAAI.

Stephen Cooper received the undergraduate degrees in chemistry and mathematics from Cornell University, Ithaca, NY, USA, and the master's and Doctoral degrees in computer science from Syracuse University, Syracuse, NY, USA. He is the Chancellor's Professor and the Director of the Jeffrey S. Raikes School of Computer Science and Management, University of Nebraska, Lincoln, NE, USA (an undergraduate honors college combining computer science and business), where he also holds an appointment as an Associate Professor with the Department of Computer Science and Engineering. He was with Computer Science Department, Stanford University, Stanford, CA, USA, where he served as an Associate Professor (teaching). His research areas lie in computer science education, with particular interest in program visualization and in trying to understand how students learn to program. He is most well known for his work with the Alice programming environment and in developing Alice-related curricular materials. He has coauthored many technical papers on Alice as well as two textbooks.