Workshop Title: ***Computer Science Student-Centered Instructional Continuum (CS-SCIC)***

Presenters:
Christine Liebe, Ph.D.
CS Education Research, Postdoctoral Fellow
Department of Computer Science
Colorado School of Mines
cliebe@mines.edu

Jane Waite, Ph.D. Cand.
Department of Computer Science
Queen's College, London
j.l.waite@qmul.ac.uk

Abstract: *Although a body of research exists defining computational thinking and confirming computational thinking as an instructional framework for teaching computer science in PreK-12 and undergraduate education, less research exists documenting instructional pathways translating computational thinking into instructional action. Little research beyond pair programming, PRIMM, and POGIL, guides computer science (CS) instructors in offers guidance on student-centered instructional strategies. In this workshop, we will share a theoretical instructional continuum offering K-20 teachers strategies that target students' needs from more to less structured CS learning and collect perceptions and input from cohorts of both higher education faculty and K-12 faculty on the Computer Science Student-Centered Instructional Continuum (CS-SCIC).*

Advertisement: *Love to teach CS but not sure why some students "get it" and others do not? Not sure how to engage the "struggling" CS students? Experience and analyze a variety of different student-centered instructional strategies in this workshop. Great for both K-12 and higher ed instructors.*

Significance and Relevance of the Topic:

Drawing on Vygotsky's zone of proximal development, CS-SCIC provides a learning continuum honoring a variety of interpersonal learning preferences. The zone of proximal learning theory is a range of understanding where students understand more than they can proficiently express and interpersonal communication is an essential aspect of learning [2]. Interpersonal learning with an instructor or through pair programming, or in a team are all effective collaborative learning experiences that prepare students for CS professions [3]. Fuller et al. [4] identified a taxonomy of learning computer science, focused on student learning, differentiating at a secondary or university level how some students may prefer to learn conceptually and others through experimentation. CS-SCIC activities provide a continuum of activities ranging from more structured to less structured experiences that can be engaged in collaboratively or independently at any time. CS-SCIC activities help teachers to recognize skills needed for independent mastery in terms of any computer science concept or programming language. Students and teachers preferring a great deal of structure may choose to progress from copy code to tinkering. Students and teachers ascribing to constructionism (Papert) [5], desirous of a less-structured approach to learning, may choose to start with tinkering and work to the left of the CS Student-Centered Instructional Continuum obtaining skills through more structured instructional practices as needed. Analysis, including qualitative input from the proposed workshop participants (IRB pending), will help inform the best application of CS-SCIC instructional strategies.

Additionally, CS-SCIC activities provide instructors with pedagogy facilitating learning advanced computing concepts and computational thinking. Students who experience all aspects of CS-SCIC gain insight into essential computer capabilities needed for advanced computing concepts, such as artificial intelligence. Computers

programmed to process "intelligently" must discern language, process data, problem-solve, and evaluate results [6]. Any CS-SCIC activity could address the development of computational thinking. According to Aho, "Abstractions called computational models are at the heart of computation and computational thinking. Computation is a process that is defined in terms of an underlying model of computation, and computational thinking is the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms." [7] Students gain background knowledge, experience, and context when they engage in copy code activities. Copying code and using scaffolded coding software helps students experience abstractions, computation, and algorithms. Modeling is an instructional activity where instructors can begin to match thought processes, metacognition, with computer code. During modeling, students begin to understand the mental inner conversation needed to create algorithms that solve computational problems.

CS-SCIC instructional strategies guide instructors in applying common CS pedagogy, such as project-based learning and tinkering in a comprehensive learning model leading towards the development of independent computer science skills. In project-based learning, students apply programming using problem-solving skills to achieve a computational solution [8]. Project-based learning is usually collaborative and helps students connect programming language and solutions contextually to their lives and the real world [9,10]. Ideally, students would demonstrate knowledge of abstract modeling and independently create functional algorithms and even programs in project-based learning. Project-based learning also offers students a chance to apply the design process, similar to the scientific process, with a stronger focus on creation, testing, and refinement. Project-based instruction can have real-world applications allowing students to learn STEM workforce skills, ethics, and become motivated via service-learning [11].
Inquiry-based instruction is more deductive than project-based learning although both instructional activities focus on problem-solving skills. Instructors use focused questions in inquiry-based instruction to guide students in discovering concepts of computer science instead of lecturing or driving instruction through structured learning. Inquiry-based instruction helps students learn what questions to ask and how to independently discover and confirm answers [12]. Inquiry-based instruction can help prepare students to learn navigate the design process required for client-based requests.

Tinkering is a creative and inventive learning experience where students explore and create computational artifacts [13, 14]. At a young age students naturally play and express creativity. Tinkering helps older students remember how to create and explore. Tinkering is completely independent learning. Instructors support through asking questions, providing resources, and offering feedback. All aspects of the instructional continuum address different types of CS learners and needs promoting more effective and accessible CS education.

Expected audience: *K-12 CS educators.*

Space and Enrollment restrictions: *We can accommodate 40 – 80 participants.*

Rough Agenda:
First hour:
10 minutes – introductions and audience discussion of instructional strategies from more structured to less structured. Audience is asked to hypothesize which students benefit from structured and which students benefit from unstructured instruction.
20 minutes – Christine Liebe and Jane Waite lead audience in examples of each instructional strategy.
25 minutes – Audience breaks up into small groups (4-5 participants) of teachers from like grades / higher ed (i.e. elementary, MS, HS, CS 0 or 1, higher CS courses). Graphic organizers depicting boxes with the following terms will be provided to each group (already do / like to do more / never considered / won't work for me / like to modify). Groups will be asked to discuss how each instructional strategy fits into any of the graphic organizer topics.
5 minutes – Question and Answer at the end.

Break: 10 min.

Second hour:
2 minutes – overview – application of the entire continuum in a semester higher ed course and in a year-long high school course, then focus groups

10 minutes – Christine Liebe presents an application of CS-SCIC activities in a high school course.
10 minutes – Christine Liebe presents an application of CS-SCIC activities in a middle school course.
33 minutes – Participants discuss utilizing the entire continuum of CS-SCIC activities in a course in either higher ed or K-12 cohorts.

Break – 10 minutes

15 minutes - Quiz – done in small groups – match student profile with best choice of instructional strategy
20 minutes – Create task – done in pairs – suggest activities related to each instructional strategy
15 minutes – Share all tasks
5 minutes - debrief, explain contact and sharing information for virtual CS-SCIC community

Third Hour
45 minutes – Small group investigation of strategies for grade levels (K-2, 3-4, 5-6, MS, HS)
15 minutes – Debrief, community share

Lunch

Fourth Hour
45 minutes – Interactive presentation about learning theories related to each CS-SCIC concept
10 minutes – Discussion and community share

Fifth Hour
30 minutes – Linking theory and student learning activities in small groups
30 minutes – Group share and demonstrations

Break – 10 minutes

Sixth Hour
15 minutes – Applying CS – SCIC to curriculum spiraling
30 minutes – Curriculum (yours or sample) analysis with a partner
15 minutes – Discussion, community share

Seventh Hour
20 minutes – preparing a short presentation for other teachers (5 slides) (can use templates provided)
20 minutes – demo slides with two other people
20 minutes – Summary, resources, next steps

Audio/Visual and Computer requirements:
Projector and Mac dongle with two hands free microphones.

References

1. Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39.
2. Tudge, J. (1992). Vygotsky, the zone of proximal development, and peer collaboration: Implications for classroom practice.
3. Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works?. *Communications of the ACM*, *56*(8), 34-36.
4. Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., ... & Thompson, E. (2007). Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bulletin*, *39*(4), 152-170.
5. Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc..
6. Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.
7. Aho, A. Computation and Computational Thinking, 2011; http://ubiquity.acm.org/article.cfm?id=1922682
8. Jaime, A., Blanco, J. M., Domínguez, C., Sánchez, A., Heras, J., & Usandizaga, I. (2016). Spiral and project-based learning with peer assessment in a computer science project management course. *Journal of Science Education and Technology*, *25*(3), 439-449.
9. Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199-237.
10. Han, S., Capraro, R., & Capraro, M. M. (2015). How science, technology, engineering, and mathematics (STEM) project-based learning (PBL) affects high, middle, and low achievers differently: The impact of student factors on

achievement. *International Journal of Science and Mathematics Education*, *13*(5), 1089-1113.
11. Payton, J., Barnes, T., Buch, K., Rorrer, A., & Zuo, H. (2015). The effects of integrating service learning into computer science: an inter-institutional longitudinal study. *Computer Science Education*, *25*(3), 311-324.
12. Lazonder, A. W., & Harmsen, R. (2016). Meta-analysis of inquiry-based learning: Effects of guidance. *Review of Educational Research*, *86*(3), 681-718.
13. Özdemir, S. (2019). Soloborative Learning: Solo Thinking, Collaborative Tinkering. *International Electronic Journal of Elementary Education*, *11*(3), 217-219.
14. Berland, M. A. T. T. H. E. W. (2016). Making, tinkering, and computational literacy. *Makeology: Makers as learners*, *2*, 196.