JCSI Journal of Computer Science Integration

Integrating Computing into Preservice Teacher Preparation Programs across the Core: Language, Mathematics, and Science

RESEARCH

LAUREN E. MARGULIEUX
PATRICK ENDERLE
PIER JUNOR CLARKE
NATALIE KING
CAROLINE SULLIVAN
MICHELLE ZOSS
JOYCE MANY

*Author affiliations can be found in the back matter of this article

ABSTRACT

This paper describes the beginning of a design-based research project for integrating computing activities in preservice teacher programs throughout a middle and secondary education department. Computing integration activities use computing tools, like programming, to support learning in non-computing disciplines. The paper begins with the motivation for integrating computing that encouraged widespread buy-in, design goals, and design parameters. The primary motivating factor for this work was preparing teachers to use technology to support learning in their classrooms. Involving computing education faculty in the preparation enabled the activities to include computer science and spread computational literacy. The paper also describes the process and year-long timeline for designing and implementing the integrations, followed by the details of the computing integrated activities. Last, the paper describes preservice teachers' reactions to computing integration, focusing on before-and-after perceptions and knowledge of computing. Preservice teachers perceptions and knowledge of computing evolved similarly to teachers who engage in different approaches to learning about integrated computing, such as in elective or educational technology courses, suggesting that this approach is effective for engaging all teachers in integrating computing. In particular, the common feature that ignited teachers' excitement about integrating computing was offering new opportunities to improve student learning and providing engaging activities within their non-computing discipline.

CORRESPONDING AUTHOR:

Lauren Margulieux

Department of Learning Sciences, Georgia State University, US

lmargulieux@gsu.edu

KEYWORDS:

computational thinking; integrated computing; preservice teacher preparation; language arts; sciences; mathematics

TO CITE THIS ARTICLE:

Margulieux, L. E., Enderle, P., Clarke, P. J., King, N., Sullivan, C., Zoss, M., & Many, J. (2022). Integrating Computing into Preservice Teacher Preparation Programs across the Core: Language, Mathematics, and Science. *Journal of Computer Science Integration,* 5(1): 1, pp. 1–16. DOI: https://doi.org/10.26716/jcsi.2022.11.15.35

Computer science is becoming a fundamental literacy for citizens and should be accessible, if not required, for all K-12 students (Bocconi et al., 2016; DeLyser et al., 2018). Similar to other literacies like language, mathematics, and science, all citizens do not need to understand advanced concepts in computer science, but they need computational literacy to succeed in their professional and personal lives (Wing, 2006; Grover & Pea, 2013). Computational literacy is important for effective cybersecurity, productive use of technology, and making creative and powerful solutions to problems (Lynch et al., 2020; Usta & Mirasyedioglu, 2021). An exemplar is children creating apps in Android App Inventor to solve problems in their schools and communities.[1]

A critical issue in the spread of computational literacy is the inequitable access to computing education in schools and communities. Women, people who are Black, Latinx, and Native American, individuals with disabilities, and those who live in low income or rural communities are underrepresented (Google & Gallup, 2016, 2017). Many resources have been devoted to programs that increase computational literacy, but they are often optional (e.g., after school programs, summer enrichment, or elective courses; Google & Gallup, 2016, 2017). Despite the efficacy of these programs for some students, they can perpetuate existing inequities because underrepresented groups are often not encouraged to participate in them or feel unwelcome (Margolis & Fisher, 2002).

An effective strategy to combat inequities in computer science is to make computing a requirement for all students, just like language, math, and science (Bocconi et al., 2016; DeLyser et al., 2018, Yadav et al., 2017). As any educator will recognize, however, finding time in the school day for required computing instruction can be an insurmountable challenge. Thus, many recent efforts have focused on integrating computing instruction in required, core courses (Yadav et al., 2014). This approach introduces all students to computing concepts and provides educators with powerful technology to use in their teaching practice (Grgurina & Yeni; 2021; Kale et al., 2018; Kong & Lai, 2021; Lynch, 2017). Thus, computing integration can improve computational literacy while providing tools for learning when it is designed to serve both disciplinary and computing goals (Guzdial, 2019; Lynch et al., 2020; Tedre et al., 2018). Our goal for the current project was to design computing integrations for preservice teacher (PST) programs that would both improve computational literacy and provide PSTs with tools, while not requiring additional instructional time to teach content within their field.

Just as educators struggle to fit computing instruction into the school day, colleges of education struggle to fit computing instruction into PST preparation programs. Some colleges of education offer instruction in computing education to non-computing PSTs through an elective course about computing integration (Yadav et al., 2016; 2017). This approach provides extended instruction about computing and how to teach it. The drawbacks, however, are that 1) all PSTs across grade bands and disciplines are taking the same course, meaning that some instruction and peer interactions might not easily map to PSTs' intended contexts, and 2) these courses are often optional, which can perpetuate inequity. Using a different approach, some universities have redesigned their required educational technology courses to include computing integration (DeLyser et al., 2018; Mouza et al. 2017; Yang & Mouza, 2021), but many universities do not require an educational technology course.

Instead of a standalone computing integration course or educational technology course, we designed computing integrated activities for disciplinary teaching methods courses. Teaching methods courses within a specific grade band and discipline introduce PSTs to pedagogical content knowledge and useful educational technologies for their specific context (Kong & Lai, 2021). This approach ensured that all of our teachers would learn computing integrated activities, increasing the number and diversity of PSTs using computing integration. Furthermore, because these activities are designed for and taught in methods courses, we were able to connect them to the pedagogical content knowledge, sociocultural, and critical aspects of learning environments (Lucarelli et al., 2021), as Kafai and Proctor (2021) call for while teaching computational literacies.

This initiative gained traction with teacher preparation faculty in part because it aligned with a year-long focus to update PSTs' preparation to use technology effectively. In the fall, six internal mini-grants of $500 each incentivized the redevelopment of courses with an increased emphasis on infusing learning technology in ways consistent with the International Society for Technology in Education (ISTE) standards for students and educators, which now includes computational thinking. The mini-grants promoted innovation and collaboration among faculty members in teacher preparation and educational technology.

Ultimately, a system was designed in which all PSTs in the middle and secondary education programs would have exposure to computing integration through their required coursework. All PSTs completed an online, hour-long module on computational thinking (CT) and computing integration. Then PSTs in English Language Arts, mathematics, and science spent a week of their methods courses learning computing integration activities that were designed between their professor and a computing education faculty member. Some of those PSTs implemented the computing integration activities with students through student teaching or practicum experiences. Though this approach has limited instruction on computing concepts, it served the goal of

introducing all PSTs to computing integration so that they are prepared to use computing activities with their classes.

Computing integration activities have been a popular area of study for at least the past five years. Thus, the primary goal of this research was to compare our system of integrating computing in PST programs, which has relatively little but targeted computing instruction, to other implementations that provide more comprehensive instruction to a broader audience. To explore the efficacy of this system, we addressed the following research questions.

1. What knowledge of computing do PSTs have after an hour of instruction on CT concepts and an hour of preparing a computing integrated activity?
2. What are the alignments that PSTs recognized between their primary disciplinary and computing that enabled them to integrate computing?

We compare the results of this system to the patterns of results from standalone computing and CT integration courses to examine its relative efficacy.

COMPUTING INTEGRATION DESIGN GOALS AND PARAMETERS

Computing integration can provide engaging learning experiences and outcomes for both computing and the primary discipline, but it must balance the goals and contexts of both fields (Hur, 2021; Kale et al., 2018; Lynch et al., 2020). The benefit of introducing integration activities in teaching methods courses is that the design caters to the specific needs of each program (i.e., the Technological Pedagogical Content Knowledge, Kong & Lai, 2021) and every PST learns the activities. The tradeoff is that the teaching methods courses are already full of content to meet the requirements for credentialing and successful teaching practice.

Based on these needs, PST programs already have strict requirements. In addition to disciplinary content, all PSTs also learn pedagogical concepts and practices. Some of these concepts and practices are required for credentialing, such as the Interstate Teacher Assessment and Support Consortium (InTASC) standards and ISTE standards. Other concepts and practices are not required for credentialing but are required to prepare PSTs to support student learning in various contexts. For example, many of our teachers will work in schools in low-income neighborhoods, which has implications for the technology that they will have access to in their classroom. These kinds of practical implications are discussed in our PSTs' teaching methods courses.

Within the context of methods classes, the level of computing knowledge that can be added to the required coursework is limited. From a computing education perspective, more instruction about computer science would be preferred, but the primary guiding factors were usefulness of concepts to PSTs' practice. Even with limited time for computing instruction, it was important to the computing education (CSEd) faculty, author Margulieux, that the activities included programming. While many computing integration activities do not require programming, it is a key tool in computing and affords the flexibility that is necessary for creative computing solutions (Guzdial, 2019). The CSEd author also argues that it is important to demystify programming for many PSTs who have little to no experience with it. Programming activities also provide immediate feedback compared to unplugged computing activities, which is beneficial as teachers explore its affordances (Šiaulys & Dagienė, 2021). Furthermore, a longitudinal study revealed that programming experience was important to retention of underrepresented groups in computing fields (Weston et al., 2019), aligning with our goal to broaden participation in computing.

Layered with computing education goals, the other programs had their own goals for this initiative. The director of preservice programs in middle and secondary education, author Sullivan, assigned the online computational thinking module as part of her course that is required for all PSTs. Her goals for including the module were to ensure that PSTs had exposure to computing, regardless of how technology was used in individual programs, and address challenges that teachers had in terms of access and availability of technology in their schools. Thus, one of the strategies in this project was to establish a framework for using technology from which PSTs could extrapolate to their classroom context. The other goal for the module was to prepare PSTs for computing integration activities in their methods courses.

Discipline-specific methods courses had unique approaches for computing integration. In the English Language Arts methods course, a primary goal was to prepare PSTs to effectively use digital tools for learning, aligned with a national goal by the National Council of Teachers of English (NCTE). NCTE's Definition of Literacy in a Digital Age positions teachers as agents of change and arbiters of technology who have the responsibility to evaluate the use of digital tools for learning (Witte et al., 2019). The position charges teachers and students to consume, curate, and create actively across contexts. The English methods course addresses this imperative by giving PSTs experience in using tools for their learning and then adapting those tools for teaching.

In the mathematics methods course, one primary goal was to prepare PSTs to use a different approach to build conceptual understandings of mathematical concepts. The

course instructor, author Junor Clarke, agrees with Buteau, Muller, Mgombelo, and Sacristan (2018) that computational thinking and computing integration is a way of empowering learners to use computers to act like mathematicians and develop 21st-century skills. She believes it is necessary and critical that education faculty and their PSTs in college courses be aware and be able to embed and implement computing in their current curricular concepts.

In the science methods courses, a primary goal was to give PSTs tools to increase engagement during instruction and to teach computational thinking, a newly included concept in science standards. The instructors, authors Enderle and King, agree that computational thinking is becoming a fundamental literacy (Bocconi et al., 2016), yet it is ill-defined as a scientific practice in the Framework for K-12 Science Education (NRC, 2012) and the Next Generation Science Standards (NGSS Lead States, 2013). Thus, another goal was to improve learning about scientific practices by engaging in computing activities that use science ideas and computational thinking. To achieve these goals, the project had the following design parameters

- Assume no prior knowledge of computing or computational thinking.
- Activities must be authentic for the primary discipline and computing.
- Activities must be accessible to PSTs and not dependent on a specific culture.
- Activities must be modifiable to fit the context of teachers' future classrooms.
- Activities must not require technology other than computer and internet.

DESIGN PROCESS

This section describes the process taken to begin this design-based research project. The implementation section highlights details for each discipline. Table 1 shows the activities with their timeline and who participated. The process took a year, which was partially dictated by when courses were offered.

The CSEd faculty member met individually with each PST faculty member to identify relevant concepts and topics. She brought general ideas to the first meeting (e.g., programming a visualization of a scientific phenomenon) to exemplify possibilities, but the discussion focused on topics in which teachers could improve their practice by using computing (e.g., students can make a dynamic visualization of a scientific phenomenon rather than a static visualization). Based on the topic identified in the first meeting, the CSEd faculty member either found existing activities for computing integration or developed them herself (details in the implementation section). Finally, both faculty members designed a lesson plan to implement the activities.

To implement the activities, the CSEd faculty member visited the teaching methods courses for one week. The primary instructor was present and helped to facilitate discussions and make connections to teaching practice in the discipline. The guest instructor introduced the motivation for computing integration, general areas within the discipline suitable for computing integration, the programming tool, and the activities. Because the instruction was given by the same person, the CSEd faculty member, fidelity of implementation was not a concern. All PSTs received the same instruction, except that the discipline-specific activity was matched to their disclipline.

INTEGRATION IMPLEMENTATION

Each of the computing integration activities is described below, including

- The programming language
  - Block-based or text-based
  - Discipline-specific or interdisciplinary
- The type of programming activity
  - Modify an existing program or creating a program
- Why the topic in each discipline was chosen

To prepare PSTs for these activities and introduce them to the contrast between consuming technological solutions

ACTIVITY	TIMELINE	PEOPLE
CSEd and PST faculty meet collectively to discuss possibilities, goals, and parameters	September	CSEd & PST faculty
Individual meetings between CSEd and PST faculty to design and develop computing integration (1–4 meetings per pair)	Spring	CSEd & PST faculty
General computational thinking module was given to all PSTs in middle and secondary programs	Summer	CSEd, PST faculty, and PSTs
Computing integration activities in the methods courses and student teaching	Fall	CSEd, PST faculty, and PSTs

Table 1 Activities Related to Computing Integration Across the Department.

and producing technological solutions through computing, the CSEd faculty and middle secondary program director, authors Margulieux and Sullivan, developed an hour-long module about computational thinking concepts and practices based on definitions of computational thinking from the literature (Aho, 2012; Armoni, 2016; Barr & Stephenson, 2011; Brennan & Resnick, 2012; Cuny, Snyder, & Wing, 2011; Denning, 2017; Grover & Pea, 2013; Tang et al., 2020; Weintrop et al., 2016; Wing, 2006, 2008). While none of the definitions from the literature match exactly, the five most common concepts among definitions are more prevalent than the others. Using this frequency distribution to identify the cut-off for inclusion, the concepts included were abstraction, algorithms, automation, decomposition, and debugging. All five concepts can be easily applied to other disciplines while including concepts that are unique to computing (i.e., automation and debugging).

In total, the CT module had six units. Besides the 5 CT concepts, we started with the motivation for computing integration and a set of three questions for thinking about computing integration. Interestingly, our set of three questions for thinking about computing closely align with Grgurina and Yeni's (2021) three steps for the CT program solving process, though they were developed separately. Their three steps are, "translating the problem under scrutiny into computational terms, constructing a computational solution, and using that computational solution in the domain," (p. 4; Grgurina & Yeni, 2021).

1. Motivation for computing integration and core questions
   a. Is a computer well-suited to help me solve this problem (or part of it)?
   b. How would I get a computer (design a system) to solve this problem?
   c. Does the computer solve the problem accurately and efficiently?
2. **Abstraction** with related concepts and practices: defining parameters, conditionals, test cases
3. **Algorithms** with related concepts and practices: mental models, logical thinking, iterative design
4. **Automation** with related concepts and practices: sequencing, humans as computers versus machines
5. **Deconstruction** with related concepts and practices: iterative design and abstraction
6. **Debugging** with related practice: rubber duck debugging

The CT module was designed to follow a general formula for each CT concept. First was a definition of the concept. The second was examples of the concept in three disciplines, rotating through a subset of English, foreign language, mathematics, science, and humanities. The third was an activity to practice applying the concept, building upon a module-long paper airplane example. Last was a reflection including questions and discussions of how the concept related to PSTs' primary discipline.

PSTs engaged with the CT module, including completing a multiple-choice quiz at the end of each unit, online during a summer course. PSTs were split into groups based on grade band and discipline and asked to discuss the concepts in a discussion board. When visiting the methods courses, the guest instructor reviewed these concepts again, focusing on examples within the discipline, before introducing the integration activities.

ENGLISH LANGUAGE ARTS ACTIVITIES

Computing integration is less common in language than in mathematics or science (Lynch et al., 2020), so fewer ideas from previous work informed the initial meeting. Most integrated computing in language revolves around digital storytelling, but it is often used as a bridge that prioritizes introducing students to computing rather than using computing to promote language learning (Kordaki & Kakavas, 2017; Parsazadeh et al., 2021). In the initial meeting, the language education faculty member, author Zoss, did not specify a topic that needed to be the focus of the activity and instead focused on computing as a tool to promote creativity and student engagement. Thus, the CSEd faculty member looked for an activity that used a block-based language to foreground the functionality of programming without requiring much pre-training of the computing tool. She also wanted to start with a sophisticated program that teachers could modify creatively to achieve different language learning objectives. The faculty members settled on a chatbot modeled on Lady Macbeth[2] developed in Pencil Code as part of Google's Exploring Computational Thinking resources.

Because the program is relatively sophisticated, PSTs first used the chatbot, asking questions to the programmed Lady Macbeth. After PSTs became acquainted with using the program, they read the code of the program to explore how the control statements were designed to produce responses based on the users' questions. Then, PSTs were asked to predict the outputs to certain questions and test those predictions (e.g., how would the program respond if you used two keywords?) to develop a better understanding of how the program worked. Progressing from using the program to modifying the program, PSTs made small changes to the program (e.g., the program recognized "hi" and "hello" as inputs, and PSTs were asked to add another greeting to this list, like "hey" in Figure 1).

```
replyto = (words) ->
  for word in words
    switch word
      when "hello", "hi", "hey"
      return "Hello. I'm waiting."

      # For example, if asked "Who are you?"
      when "name", "who"
      return "Lady Macbeth."

      # E.g.: "Have you heard about the murder?"
      when "power", "kill", "duncan", "murder", "king"
      return """
        I love power.
        I will do anything for power.
      """
```

Figure 1 A Segment of the Program Used to Create a Chatbot in an English Language Arts Activity.

Some of these changes could be completed in the block-based interface, but others required the PSTs to access the text-based interface to make changes. Asking the PSTs, through a scaffolded task, to change the program in the text-based interface gave them the skills to customize the original program in creative ways. To reinforce the extent to which they could modify the program to serve their needs, the instructor discussed how they could change the program to become a dictionary of literary devices. Users could ask about a literary device, and the program would output a definition and example. The PSTs then had time outside of class to modify the chatbot for a literary text that they used in a long-term planning activity.

MATHEMATICS ACTIVITIES

For the secondary mathematics teaching methods course, the CSEd and Math faculty decided to introduce PSTs to a well-established and popular curriculum for integrating algebra and computing called Bootstrap.[3] The Bootstrap curriculum for Algebra has been refined through many iterations based on data collected in authentic classroom environments (Schanzer et al., 2018). It has learning objectives connected to national standards in mathematics (Common Core) and computer science (Computer Science Teachers' Association). The programming language is a text-based language that can be used to solve problems in multiple disciplines.

The evidence-based curriculum is free to use, and one goal for the integration was to introduce PSTs to the curriculum and make them comfortable enough to use it in their algebra courses. It takes about 25 hours to complete, so one methods course was not sufficient to work through the entire curriculum. Instead, PSTs were introduced to the first of ten units, emphasizing connections to a conceptual understanding of mathematics and how the computing component allows students to explore different representations and applications of mathematical functions. Then the instructor introduced PSTs to the remaining resources available and discussed how to engage with each component.

Unlike the other implementations, the primary instructor of the course, author Junor Clarke, lead the computing integration class while the CSEd faculty provided support. Thus, this integration made many more connections throughout the class to how the PSTs could use this in their classroom at a more granular level. For example, the Bootstrap curriculum provides lesson plans for each unit, and Dr. Junor Clarke demonstrated activities that teachers could use to complete each segment of the lesson plan. In the other implementations, the activities were taught only at the lesson plan level, and PSTs would need to decide how each segment of the lesson plan would be implemented. The approach that Dr. Junor Clarke took is likely valuable to those in her course, and she spent a considerable amount of time exploring and learning the Bootstrap curriculum to be able to teach it independently.

SCIENCE ACTIVITIES

Faculty members designed two integrations with science, one for middle school and another for high school. Science education faculty, authors Enderle and King, engaged in both integrations. In the initial meeting for middle school, the following design specifications were set: 1) use an interactive scientific model of a phenomenon to help visualize it, and 2) use an interdisciplinary programming language for activities because middle school PSTs specialize in at least two subjects. Modeling and simulations to visualize scientific phenomena is a common area to connect science with computing (Grgurina & Yeni, 2021; Weintrop et al., 2016). However, many of these tools foreground the visualization and do not automatically show the program that creates it, e.g., PhET and NetLogo, which does not serve our goals of teaching programming. The CSEd faculty recommended a block-based language so that PSTs could use advanced concepts to create visualizations without learning complex syntax. The team chose Pencil Code to meet the design specifications because it is block-based and uses turtle graphics. Turtle graphics, a paradigm in which the programmer tells a digital turtle how to move around a space and often includes drawings or other visual components, were deemed appropriate for creating visualizations.

Based on the timeframe of the course, the selected topic was electromagnetic waves. Therefore, the CSEd faculty created an activity using Pencil Code for modeling electromagnetic waves. To introduce the activity, Dr. King guided the PSTs through a review of relevant scientific concepts for waves, in which PSTs defined the components of a wave and drew waves on whiteboards. After a tutorial of Pencil Code, the guest instructor walked PSTs through a simplified wave model with only wavelength and not

amplitude. For homework PSTs were asked to expand on their models so that they included amplitude (see Figure 2).

The Pencil Code models were designed to include variables. The variables supported easy manipulation of values in the model and, at Science ed faculty member's request, the use of equations to connect to mathematical concepts. The models also included functions and for loops at the CSEd faculty member's request to include more computing concepts. Though a more elegant program might include the wavelength and amplitude as parameters of the wave function, this design better suited the project's goals and simplified the concept of functions by excluding parameters.

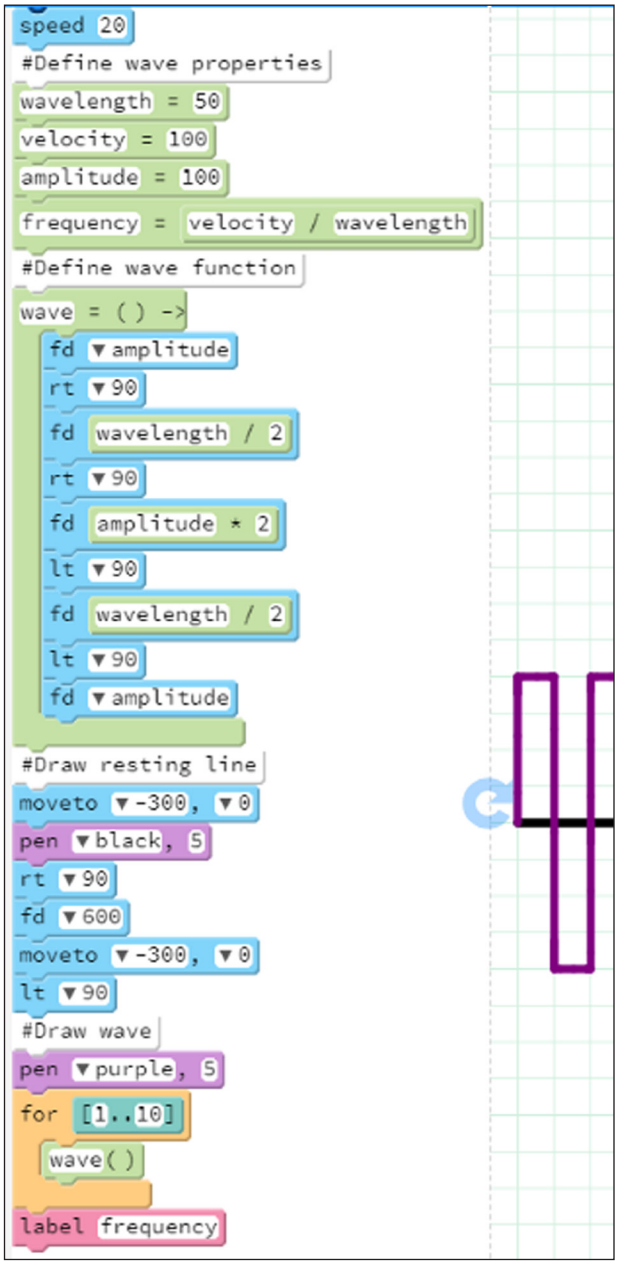

Figure 2 Program Used to Create Wave Visualization in Science Activity.

The activity for the secondary science methods course was quite different. It was implemented near the end of the semester, and there was no particular science topic that needed to be included. Instead, the team opted to introduce the PSTs to NetLogo and provide PSTs autonomy to choose a topic. NetLogo is a text-based language originally designed for creating simulations in science, but it has expanded to other disciplines. NetLogo also offers an expansive collection of pre-made simulations, which were used as the basis for the activity. After introducing the PSTs to NetLogo and its library of simulations, instructors asked them to pick a simulation that they could use in their classrooms. Once PSTs selected a topic, instructors asked them to access the source code and try to match the code to its functionality in the simulation. PSTs then read through the code and modified a piece of it (e.g., delete a graph or change the output of something) and test whether their change worked to change the simulation as desired. The CSEd and Science faculty helped PSTs as they read and modified the code. The goal of this activity was to share the NetLogo simulations for PSTs to use. Furthermore, the activity was meant to empower PSTs to modify pre-built simulations to work better for their classroom and to show them that they, and their students, could understand and modify code to make it work for them, even if they did not have the skill to write it independently.

METHOD

To address the research questions and evaluate the effect on PSTs of the CT instruction and computing integration activity in the classroom and in student teaching, researchers collected qualitative and quantitative data about the process and outcomes of the intervention. The project explored the effect of the intervention on PSTs' knowledge and beliefs at three points: before CT instruction, immediately after CT instruction, and after instruction about a specific computing integration activity in their methods course. Some cohorts also include a fourth data point, after student teaching or practicum.

Because this project is an early-stage, design-based research project, its primary objective is to identify, bottom-up, variables and themes that are important to the design of CT instruction across PST programs. As such, it does not yet employ experimental or intervention methods, which would be appropriate to examine the effect of variables and themes in the next stage of design-based research. While we have a relatively large sample size for this stage of the project, our results cannot be interpreted within a cause-and-effect paradigm because we do not have strong *a priori* expectation of either cause or effect. The purpose of this work to develop a basis for *a priori* hypotheses that can be tested later.

PSTS, INSTRUCTORS, AND INSTITUTIONS

Across methods courses, 99 PSTs who participated in the study had complete datasets and were included in the analysis (see Table 2 for demographic information).

The instructors of the methods courses specialize in education within their discipline and teacher preparation. The CSEd faculty had little background with teacher preparation in language, mathematics, or science. Our approach combined areas of expertise using a co-teaching model with the PSTs. For student teaching, we partnered with local public schools in Atlanta, Georgia, which primarily serve groups that are underrepresented in computing.

DATA COLLECTION SOURCES AND ANALYSIS

Throughout the project, we collected data from all PSTs through a CT survey. The CT survey was based on the survey from Yadav et al. (2014; see Appendix A for full pre- and post-surveys). It was completed before the CT instruction (pre-survey in July), at the end of CT instruction approximately an hour later (post-survey in July), and at the end of the methods course (post-post-survey in September). The survey had a quantitative component that asked PSTs to rate on a 5-point scale 1) their familiarity with CT, 2) how easily CT can be integrated into other subjects, 3) how comfortable they would be integrating CT, and 4) their general comfort with using computers. The qualitative component of the survey asked PSTs to explain/define 1) CT, 2) how they might implement CT in their class, 3) barriers that they might face implementing CT, and 4) list three things that someone who knows computing could do.

To supplement this data source, the CSEd faculty member took notes during the methods course and student teaching. Because she was engaged in instruction, these are not systematic field notes from an impartial observer. Instead, the notes recorded the topics discussed during instruction and anything unexpected that occurred to inform future design. Additional sources of data that not all PSTs completed include reflections from PSTs during the CT module, assignments to adapt the computing integration activity introduced in the methods course, and post-student-teaching reflections. Data from these additional sources are used only to provide context for the primary data analysis.

To analyze quantitative data and quantitative coding, we used only descriptive statistics. Inferential generalizations were not a goal at this early stage of design-based research, and the quantitative data collected was minimal. Thus, most data analyses focused on qualitative data using content analysis (Hsieh & Shannon, 2005) with NVivo 12 software. Content analysis allowed themes to emerge from the data by iteratively coding the data to explore different interpretations (see Table 3 for codes created at each round of analysis). Each component of the data

PST race	White = 40	African American or Black = 38	Latinx = 10	Other or multiracial = 11
PST gender	Women = 56	Men = 38	Transgender or nonbinary = 5	

Table 2 Demographic Characteristics of PSTs Participants.

Initial Node	RQ1: What knowledge of computing do PSTs have after an hour of instruction on CT concepts and an hour of preparing a computing integrated activity?	
First Round	CT Definitions	Expressions of Uncertainty
Second Round	• Using technology • Problem solving or thought process • Listing concepts introduced in module • Using computers to solve problems • Decomposition	No additional nodes identified
Third Round	No new nodes identified	
Initial Node	RQ2: What are the alignments that PSTs recognized between their primary disciplinary and computing that enabled them to integrate computing?	
First Round	Effective Practices for Teaching CT	
Second Round	• Using computers or CT as a tool for learning • Teaching the computer as a way to learn • Using the program to understand student thought processes • Using examples that are personally meaningful	
Third Round	No new nodes identified	

Table 3 Qualitative Codes Created in Each Round of Analysis.

collection sources was coded, and one component could be coded into multiple nodes.

The initial, exploratory nodes were based on which research question the data addressed. During the first round of analysis, researchers classified components into the research question nodes and made additional nodes for high-level themes within each research question. For the first research question, additional nodes were added for CT definitions and expressions of uncertainty. For the second research question, a node was added for effective practices for teaching CT to preservice teachers. During the second round, researchers classified components within each research question into the high-level themes and made additional nodes for sub-themes, which are described in Table 3 and the results. During the third round, researchers classified components within the themes into sub-themes and did not recognize additional thematic nodes. All content was scored by two raters after the coding scheme was finalized. Initial inter-rater reliability was 82% agreement based on coding PST responses as including or excluding a node (i.e., total agreement based on binary judgments). Disagreements were resolved through discussion.

RESULTS

The results section is organized around the research question and the qualitative nodes identified through content analysis. The qualitative nodes are supplemented with descriptive quantitative data when applicable. First is research question 1: What knowledge of computing do PSTs have after an hour of instruction on CT concepts and an hour of preparing a computing integrated activity?

EVOLUTION OF CT DEFINITIONS

We examined PSTs' definitions of CT across multiple time points to examine how their understanding evolved as they engaged with computing. The first definitions they provided were on the CT survey before any instruction. On the pre-survey, 42 of 99 PSTs said that they definitely had not heard of CT before or might have heard of it. Only 16 had definitely heard of it, but some of their definitions were of computing in a mathematics context rather than a computer science context. Overall, the PSTs offered a wide range of definitions of CT with the most common responses being

- Using technology (26 PSTs) - "Incorporating computer skills and technology into classroom settings and lessons"
- Problem solving or thought process (39 PSTs) - "Possibly a logical, methodological type of thought process"

On the post-survey at the end of the CT module, PSTs gave more technically correct answers that closely mirrored instruction,

- Listing concepts introduced in module (62 PSTs) - "Computational thinking is applying automations, algorithms, decomposition, and debugging to solve problems"
- Using computers to solve problems (23 PSTs) - "Problem solving methods through computers"

The same survey was given at the end of the instruction in PSTs' methods course that focused on a specific activity for integrating computing into their primary discipline. The definitions still focused on the five CT concepts (i.e., abstraction, algorithms, automation, deconstruction, and debugging) discussed in class, but they expanded to include applicability of CT within their discipline while using the computer as a tool. Most PSTs (n = 54) gave an answer like,

> "CT is a problem-solving method that emulates computational patterns and behaviors such as how a computer would solve the problem. CT is broken into five parts to make the process easier to understand and make it easier to walk through to find a solution to the problem. CT is usually implemented when creating and executing computer programs however it has been found useful across disciplines including mathematics, science, and the humanities. CT is also good on eliminating whether or not a computer would be useful to solve the problem or not. It encourages problem solving and increases digital literacy."

These responses were sometimes not entirely accurate, but they represented an introductory, simple understanding of the concepts rather than systematic misconceptions.

A third of PSTs (n = 31) repeated the three questions used to define CT in class with minimal, but thoughtful, additions.

> "Thinking computationally basically means answering three questions: Should a computer help me solve this problem? How would I get a computer to solve this problem? Does the computer solve the problem accurately and efficiently? It is useful because the steps involved with CT help students to 'keep working' or 'keep trying' to solve a problem."

The middle school science PSTs (n = 11) completed the survey again after student teaching. They provided definitions of CT that were less reliant on using computers

or listing CT concepts. Their understanding seemed to have evolved to subject-independent process for solving problems that could be enhanced with the use of computers. It is important to note that middle school PSTs specialize in two areas, so they were likely taking another methods course concurrently. The common themes were

- Decomposition (4 PSTs) - "CT allows students to break things down and understand better of the little things that make up the bigger things."
- Problem solving or thought process (3 PSTs) - "A step-by-step process of learning and educating in which you approach problems and information from an analytical point of view"
- Using computers to solve problems (5 PSTs) - "You can use computational thinking in every subject to show how computers can be used to do more complex problems, or repeated sets of data."

As expected, PSTs definitions of CT evolved as they gained experience with CT concepts and computing integrated activities. The general GT module helped to improve the factual accuracy of PSTs definitions of CT. After learning the computing integrated activity, and especially after practicing it in student teaching, PSTs definitions began to become more conceptual in nature, though they still included some factual inaccuracies.

EXPRESSIONS OF UNCERTAINTY

PSTs expressed doubts in their ability to implement a computing integration activity during the intervention but gained confidence through instruction. From the pre-survey to the post-survey to the post-post-survey on the question, "I am confident that I could integrate computational thinking into my future classroom," with a scale of *1 - Not at all* to *5 - Completely*, confidence grew evenly from 3.55 to 3.85 to 4.07. In contrast, in response to the question, "Can computational thinking be integrated into non-computing classes?" with a scale of *1 - Not practically* to *4 - Easily*, scores increased, but not linearly. The average increased from 3.04 before the CT module to 3.38 after the CT module. Scores increased only slightly after the methods courses to 3.43. For the PSTs who engage in student teaching, however, all but one PST gave the highest rating, including those who originally said CT could not practically be integrated. Because these ratings were not used in an experimental paradigm, they describe only the PSTs in the study and should not be generalized.

During the CT module, PSTs' responses to reflection questions suggested that they easily understood the five CT concepts in relation to their prior knowledge about their discipline. When introduced to the programming environments during computing integration activities, the PSTs were immediately active, with many starting to try things before formal instruction began. While PSTs might be uncertain at the beginning of instruction, they tended to engage quickly in the activity and thoughtfully apply prior knowledge to this new tool.

Based on the CSEd faculty member's field notes during student teaching, before class began many PSTs were uncertain and nervous about the computing integrated activities. They were motivated, however, by wanting to provide hands-on learning activities for their students and multiple methods of learning content. PSTs seemed much more comfortable using the base pre-existing program that the CSEd faculty member provided as a starting point than the adapted programs that they had developed through their methods class. PSTs who used the base program focused more on disciplinary learning objectives, like they would with any pedagogical tool, while those who used adapted programs felt unsure about the validity or accuracy of their program and focused more on computing learning objectives. In a post-student-teaching reflection, most PSTs said they would be excited to use a computing integration activity in their class as long as someone else had already developed the program or algorithm.

The positive reactions of the students during student teaching motivated the PSTs to continue computing integration. Many of the students had little experience with computing or programming. Like the PSTs, students immediately began trying out features when they opened the programming environment. In one exchange, a student said, "It's a lot of testing, that's the cool thing about this." In the middle school science class, students created their own models and compared them. During the comparisons, students said, "How did you do that?!" "I didn't know I could do this," and "There's different parts of science, and [computing] is one of them." In a post-student-teaching reflection, many PSTs mentioned the excitement of the students as a motivation to include a computing integration activity in their future classroom, such as, "We asked the students if they'd want to do something like this in the future and all their hands shot up!"

EFFECTIVE PRACTICES FOR TEACHING CT TO PSTS

Effective practices for teaching PSTs was the focus of the analysis of the second research question: What are the alignments that PSTs recognized between their primary disciplinary and computing that enabled them to integrate computing?

PSTs were asked to reflect on CT concepts during the general, hour-long CT module and discuss what resonated most with them. The purpose of this question was to identify parts of the instruction PSTs found most interesting

or applicable to replicate or expand in the future. The PSTs gave varied response to this question, though, suggesting that PSTs resonated with different aspects of the instruction.

One general theme for some PSTs was using computers or CT as a tool for learning (n = 32). This theme aligns with the primary design goal of the project to provide a new tool for teachers to use in their classrooms. For some PSTs, CT was a new way of problem solving,

> "I learned how to problem solve using different methods, especially in ways that a computer would execute. The part that resonated with me the most is the rubber duck [debugging]. When we hit a roadblock in problem-solving, we can explain our problems and it helps to decode our goal."

PSTs also identified the content they could teach with CT, including computing concepts, "I never learned how to code as a student, so therefore I think teaching my students will be a great experience," or teaching students about their discipline with computing, "It was exciting to learn a new way to relate scientific ideas to my future students with technology they may not be introduced to yet."

Another general theme emphasized "teaching" the computer as a way to learn (n = 43). This theme aligns with part of the instruction in the methods course that compares writing a program to teaching a person. This part of instruction explicitly stated that students could teach concepts to the computer by writing a program.

> "It really resonated with me that teaching is a good way to learn…I can most certainly attest to teaching being a great way to learn, and CT is just that: explaining your processing down to the most finite details so you know exactly what's going on."

In a related theme, a smaller group saw the program as a tool to understand student thought processes (n = 17). This theme also aligns with part of the instruction in the methods course that by writing a program, students are formalizing and externalizing their thought process, which might help teachers or peers understand what piece is missing when students are stuck.

> "It also helps me as a teacher because I can ask a student how they went through it and got the answers that they did. Being able to reflect on each step and figure out the student got from A to Z is very useful to student and teacher."

Based on field notes, this theme might have been less popular than teaching the computer as a way to learn because many teachers were not confident in their ability to understand students' code. However, because this explanation is from the researchers' perspective, it would require more research to determine its validity from the PSTs' perspective.

The last general theme was PSTs appreciated the use of examples that were personally meaningful to them (n = 19). The examples could be relevant to them as a teacher, "What really resonated with me was the way it was so accessible. It introduced an entirely new concept by making it relate to the things that are most important to us; reaching our future students!" or to everyday life, "It used real world issue to help us understand the concepts." This theme was also less common than the others, but it was also not explicitly part of the design, except that it follows fundamental instructional design.

LIMITATIONS

As design-based research in the early stages, the primary goal of research at this point is to identify dimensions and concepts that are relevant to measure in future work. As a result, we cannot responsibly generalize any of our findings beyond the PSTs with whom we worked because that was not the goal of our measures. Instead, we have identified themes in primarily qualitative data from a sample of 99 PSTs across academic disciplines to better understand and research this new paradigm of integrated computing education. The themes identified from these results of this early-stage design-based research project will be explored more directly and measured more explicitly in future work.

DISCUSSION

From a CSEd perspective, the primary motivator for this project is to broaden participation in computing by integrating computing into the teaching practices of other academic disciplines and, thus, give students experience computing. Addressing this broad goal required designing instruction for PSTs within the constraints and requirements of their preparation programs. Thus, the focus of our design-based research was to explore how our instruction 1) achieved CS/CT learning objectives given the constraints for CS/CT instruction (i.e., RQ1 about knowledge of computing given limited instruction), and 2) afforded PSTs to teach computing within the context of their primary discipline(s) (i.e., RQ2 about alignment between their primary discipline and computing). In general, the results from this stage of the project found that integrating computing instruction into teaching methods courses, even though it greatly restrained

the amount of instruction, was successful for achieving learning objectives for both CS/CT and the primary discipline.

The results from this study are similar to those from other studies about teacher preparation for integrated computing that used different approaches, such as elective courses (e.g., Yadav et al., 2014, 2017) or educational technology courses (e.g., Yang & Mouza, 2021). In these different approaches, PSTs typically receive at least two weeks of CS/CT instruction (Kong & Lai, 2021; Rodrigues da Silva et al., 2020; Yadav et al., 2014, 2017; Yang & Mouza, 2021) rather than the two hours our PSTs received. Despite less instruction, our PSTs developed increasingly accurate, though not conceptual, knowledge of CT, just like teachers from other studies (Kong & Lai, 2021; Mouza et al., 2017; Rodrigues da Silva et al., 2020; Yadav et al., 2014, 2016). Just like teachers from other studies, our PSTs also improved their confidence in teaching computing but were still fairly uncertain in their ability to implement activities independently in their classrooms (Kong et al., 2020; Rich et al., 2017; Yadav et al., 2014, 2016, 2021). These similarities suggest that our more restricted and highly targeted approach to computing integration in methods courses, i.e., designed for teachers within a specific discipline and within a specific grade band, does not produce worse outcomes than more general CT/CS instruction.

The gold standard for computing integration instruction would be that teachers are able to adapt or create computing integration activities for the needs of their students and classroom. By this standard, our approach also performs similarly to other approaches, which is that it fails. Multiple teacher educators have successfully enabled teachers to implement pre-designed integrated computing activities in their classroom by emphasizing CT concepts and providing CS instruction in the context of a specific integration activity (e.g., Kong & Lai, 2021; Margulieux & Yadav, 2020; Rodrigues da Silva et al., 2020). However, these approaches alone, without significant ongoing professional development, classroom experience, and online learning communities, do not provide teachers with the skillset and self-efficacy to design and implement their own computing integration activities (Hew & Hara, 2007; Rich et al., 2017; Yadav et al., 2021). In contrast, Kong et al. (2020) found that in order to reach this standard, teachers needed two full college courses that included sustained programming instruction, equivalent to the introductory programming instruction computer science undergraduates receive. Whether this standard is attainable in the future, it is not currently reasonable within PST programs in the current study. Thus, similar outcomes with other CT/CS integration approaches is sufficient to justify the pursuit of this methods-course-based approach.

The success of our approach is also notable in the context of the diversity of PSTs in our study. Part of our motivation for computing integration is to teaching computing to teachers, and eventually their students, who are from groups that are underrepresented in computing. The PSTs who we worked with are predominately from groups underrepresented in computing, whether based on race (48% Black, African American, or Latino/a/x) or gender (57% women). While our research does not include racial or gender comparisons, it is important to recognize that our successful PSTs included those from underrepresented groups. Further, when we observed student teaching with students who were from the same underrepresented groups, the students were highly engaged with the computing integration activity. While this project is about broadening participation generally, rather than addressing racial or gender diversity specifically, we found no evidence our mandatory computing instruction was less effective than elective computing instruction, which tends to have less diversity of PSTs (Bocconi et al., 2016; DeLyser et al., 2018, Yadav et al., 2017).

KEY DESIGN RESOURCES

To reach this stage of the design project, we needed buy-in from each stakeholder. The computing education faculty wanted to spread computational literacy; the disciplinary education faculty wanted to provide technological tools to support teaching and learning; and administration wanted to update the use of technology in programs. Because the motivations for each contributor were different, spending time clarifying goals at the beginning of the process was critical. After we agreed upon goals and an approach to reach them, we could return to them throughout the design process to guide decisions. This project was also supported by the administration, who facilitated discussions between areas that less frequently work together and incentivized the work through the internal mini-grant program.

When designing the computing integration activities, a few resources were key. First, it was important to start with existing, well-designed computing integration activities. For the mathematics activity, we were able to adopt an existing activity as is. For the English activity, we started with a chatbot program and adapted it to serve our needs. This modification highlights the second key resource, someone who can modify or create computing integration activities. While many integration activities already exist, if the activity needs to be about a specific topic, like waves in science, creating an activity is useful. The computing education faculty member who created activities is not a computer scientist and has very little experience with programming. It was sufficient with a low level of programming skill to invest time figuring out how to implement the activities, much like the teachers might do in their classrooms.

The last key resource was the pedagogical expertise of the education faculty. Once we selected or created activities,

the education faculty designed the lesson plans that ensured activities helped achieve the learning objectives within their disciplines. Their expertise allowed us to embed computing and computational thinking within an authentic learning environment that connected to teachers' prior knowledge and identities. This kind of situated computational literacy is more sustainable and relevant across different areas of education (Kafai & Proctor, 2021). In addition, the computing learning objectives revolved around PSTs learning to use and modify the integration activities. As with much of computing, the real power comes from the adaptability and creativity that it affords. Therefore, our goals were to enable PSTs to flexibly use computing integration activities in their classrooms to provide a powerful tool for learning while spreading computational literacy to all students.

KEY TAKEAWAYS

The key takeaways that we learned from this early-stage design-based research, which we will carry over to future work, were

- PSTs' knowledge: PSTs' knowledge evolved rapidly through only a few hours of instruction. While they could make modifications to computing integrated activities by the end of their methods class, they were still most comfortable using pre-designed activities with students.
- Faculty and PST motivation:
  - Many education faculty and PSTs resonated with the purpose of using programming as a tool to learn by teaching the computer. They found this valuable for learners from a metacognitive perspective to test their knowledge and for teachers to be able to see how learners were thinking, by formalizing and externalizing their knowledge through creating a program.
  - PSTs who used activities in student teaching also liked how the activities created an interactive learning environment, both for the students to work independently while receiving immediate feedback from the computer and to work with their peers while sharing digital artifacts created with computing6.
- Programming education standard that did not carryover: Programming education uses pseudocode, code-like plain language, to conceptually represent programming concepts without accurately writing correct syntax, and it is often used as intermediate step to writing code. We tried to use pseudocode in this way during instruction, but PSTs were unfamiliar with code paradigms. As a result, trying to write pseudocode was more confusing than using block-based code or reading text-based code.

The first two takeaways regarding PSTs' knowledge and motivation align with other findings from the literature. Just as we found that PSTs' knowledge evolved faster than their comfort in using activities in the classroom, Hur (2021) also found that PSTs hesitated to teach computing in classrooms, even though their confidence and interest in computing rapidly increased through a multi-week unit on computer science. Furthermore, some of the key takeaways align with key pedagogical supports that Jocius et al. (2021) found to help middle and high school teachers be successful in integrating CT into their classrooms in other disciplines. They identified "articulating a key purpose for CT infusion, scaffolding, and student collaboration," (p. 175, Jocius et al., 2021), which aligns with using programming as a metacognitive tool, providing pre-designed activities, and using programming as an interactive tool, respectively.

The final takeaway, however, we believe is a unique contribution. Few academic programs teach programming through computing integration activities rather than through more traditional computer-science-focused programming instruction. As we found, it is likely that not all instructional techniques that work in standalone programming education will work for learning programming through computer integrations. Designers should be critical in the use of these techniques, even if they are common and effective in programming education.

In this paper, we have presented the early stages of a design-based research project to evaluate the methods and techniques our college of education is using to integrate computing throughout our middle and secondary school PST programs. We have highlighted important decisions and discussions that we built our design around to fulfill the goals of all stakeholders. In addition, we detailed activities that we created and how we implemented them with our PSTs. From this implementation, we presented qualitative and descriptive quantitative data to illuminate the experience of our PSTs as they learned about computing integration and how it can be used in their classroom. Finally, we reflect on key resources that made this project successful so far and key takeaways that can be used to continue this work at our university and others.

NOTES

[1] appinventor.mit.edu/explore/blog.

[2] bit.ly/CTchatbot.

[3] https://www.bootstrapworld.org/materials/algebra/.

ADDITIONAL FILE

The additional file for this article can be found as follows:

- **Appendix A.** CT Survey – Pre-module and Post-class. DOI: https://doi.org/10.26716/jcsi.2022.11.15.35.s1

ACKNOWLEDGEMENTS

This work is funded in part by the National Science Foundation under grant #1941642.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR CONTRIBUTIONS

Author Margulieux was responsible for creating a first draft of the paper, combining contributions from other authors, and editing throughout. The other authors were primarily responsible for contributing to the sections relevant to their disciplines within the project and providing edits on the full draft.

AUTHOR AFFILIATIONS

Lauren E. Margulieux orcid.org/0000-0002-8800-2398
Department of Learning Sciences, Georgia State University, US
Patrick Enderle orcid.org/0000-0002-3018-4416
Department of Middle and Secondary Education, Georgia State University, US
Pier Junor Clarke orcid.org/0000-0002-7432-1881
Department of Middle and Secondary Education, Georgia State University, US
Natalie King orcid.org/0000-0002-4465-1409
Department of Middle and Secondary Education, Georgia State University, US
Caroline Sullivan orcid.org/0000-0001-6442-2868
Department of Middle and Secondary Education, Georgia State University, US
Michelle Zoss orcid.org/0000-0002-2183-1300
Department of Middle and Secondary Education, Georgia State University, US
Joyce Many orcid.org/0000-0002-7194-2973
Dean's Office, Georgia State University, US

REFERENCES

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835. DOI: https://doi.org/10.1093/comjnl/bxs074

Armoni, M. (2016). Computer science, computational thinking, programming, coding: The anomalies of transitivity in K-12 computer science education. *ACM Inroads*, *7*(4), 24–27. DOI: https://doi.org/10.1145/3011071

Barr, V., & **Stephenson, C.** (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48–54. DOI: https://doi.org/10.1145/1929887.1929905

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., & **Punie, Y.** (2016). Developing computational thinking in compulsory education. *European Commission, JRC Science for Policy Report*, 68 pp.

Brennan, K., & **Resnick, M.** (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 Annual Meeting of the AERA*, Vancouver, Canada (Vol. 1, p. 25).

Buteau, C., Muller, E., Mgombelo, J., & **Sacristan, A. I.** (2018). Computational thinking in university mathematics education: A theoretical framework. Retrieved from http://sigmaa.maa.org/rume/crume2018/ Abstracts_Files/Submissions/107_Computational_Thinking_in_University_Mathematics_Education__A_Theoretical_Framework.pdf

Cuny, J., Snyder, L., & **Wing, J. M.** (2011). Computational thinking—what and why? https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

DeLyser, L. A., Goode, J., Guzdial, M., Kafai, Y., & **Yadav, A.** (2018). Priming the computer science teacher pump: Integrating computer science education into schools of education. Report published by CSforAll (pp. 1–62).

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33–39. DOI: https://doi.org/10.1145/2998438

Google & **Gallup.** (2016). Diversity gaps in computer science: Exploring the underrepresentation of girls, Blacks and Hispanics. Retrieved from http://goo.gl/PG34aH

Google & **Gallup.** (2017). Computer science learning: Closing the gap: Rural and small-town school distrticts. Retrieved from http://services.google.com/fh/files/misc/computer-science-learning-closing-the-gap-rural-small-town-brief.pdf

Grgurina, N., & **Yeni, S.** (2021, November). Computational thinking in context across curriculum: Students' and teachers' perspectives. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (pp. 3–15). Cham: Springer. DOI: https://doi.org/10.1007/978-3-030-90228-5_1

Grover, S., & **Pea, R.** (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43. DOI: https://doi.org/10.3102/0013189X12463051

Guzdial, M. (2019). Computing education as a foundation for 21st century literacy. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 502–503). ACM. DOI: https://doi.org/10.1145/3287324.3290953

Hew, K. F., & **Hara, N.** (2007). Empirical study of motivators and barriers of teacher online knowledge sharing. *Educational Technology Research and Development*, *55*(6), 573–595. DOI: https://doi.org/10.1007/s11423-007-9049-2

Hsieh, H. F., & **Shannon, S. E.** (2005). Three approaches to qualitative content analysis. *Qualitative health*

research, *15*(9), 1277–1288. DOI: https://doi.org/10.1177/1049732305276687

Hur, J. W. (2021). Pre-service teachers' beliefs, confidence, and interest in computer science education. In C. Mouza, A. Yadav, & A. Ottenbreit-Leftwich (Eds.), *Preparing Pre-Service Teachers to Teach Computer Science* (pp. 29–48). Information Age Publishing, Inc.

Jocius, R., Joshi, D., Albert, J., Barnes, T., Robinson, R., Cateté, V., ... & **Andrews, A.** (2021, March). The virtual pivot: Transitioning computational thinking PD for middle and high school content area teachers. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (pp. 1198–1204). DOI: https://doi.org/10.1145/3408877.3432558

Kafai, Y. B., & **Proctor, C.** (2021). A revaluation of computational thinking in K–12 education: Moving toward computational literacies. *Educational Researcher*, 6 pp. DOI: https://doi.org/10.3102/0013189X211057904

Kale, U., Akcaoglu, M., Cullen, T., Goh, D., Devine, L., Calvert, N., & **Grise, K.** (2018). Computational what? Relating computational thinking to teaching. *TechTrends, 62*, 574–584. DOI: https://doi.org/10.1007/s11528-018-0290-9

Kong, S. C., & **Lai, M.** (2021). A proposed computational thinking teacher development framework for K-12 guided by the TPACK model. *Journal of Computers in Education*, 1–24. DOI: https://doi.org/10.1007/s40692-021-00207-7

Kong, S. C., Lai, M., & **Sun, D.** (2020). Teacher development in computational thinking: Design and learning outcomes of programming concepts, practices and pedagogy. *Computers & Education*, 151, 103872. DOI: https://doi.org/10.1016/j.compedu.2020.103872

Kordaki, M., & **Kakavas, P.** (2017). Digital storytelling as an effective framework for the development of computational thinking skills. *EDULEARN2017*, 3–5. DOI: https://doi.org/10.21125/edulearn.2017.2435

Lynch, T. L. (2017). How English Teachers Will Save the Future: Reimagining Computer Science as the Language Art It Is. *Journal of English Teaching through Movies and Media*, *18*(4), 163–180. DOI: https://doi.org/10.16875/stem.2017.18.4.163

Lynch, T. L., Ardito, G., & **Amendola, P.** (2020). *Integrating Computer Science Across the Core: Strategies for K–12 Districts*. Eye on Education. DOI: https://doi.org/10.4324/9780429243783

Lucarelli, C., Long, J., Rosato, J., Treichel, C., & **Benedict, H.** (2021). Creating change agents: A teacher preparation model that prepares all teachers to facilitate computer science concepts. In C. Mouza, A. Yadav, & A. Ottenbreit-Leftwich (Eds.), *Preparing Pre-Service Teachers to Teach Computer Science* (pp. 91–112). Information Age Publishing, Inc.

Margolis, J., & **Fisher, A.** (2002). *Unlocking the clubhouse: Women in computing*. MIT press.

Margulieux, L., & **Yadav, A.** (2020). Middle Science Computing Integration with Preservice Teachers. In *Society for Information Technology & Teacher Education International Conference* (pp. 63–72). Association for the Advancement of Computing in Education (AACE).

Mouza, C., Yang, H., Pan, Y. C., Ozden, S. Y., & **Pollock, L.** (2017). Resetting educational technology coursework for pre-service teachers: A computational thinking approach to the development of technological pedagogical content knowledge (TPACK). *Australasian Journal of Educational Technology*, *33*(3). DOI: https://doi.org/10.14742/ajet.3521

NGSS Lead States. (2013). Next Generation Science Standards: For States, By States. Washington, DC: The National Academies Press.

National Research Council. (2012). A framework for K-12 science education: Practices, crosscutting concepts, and core ideas. *Committee on a Conceptual Framework for New K-12 Science Standards. Board on Science Education, Division of Behavioral and Social Sciences and Education*. Washington, DC: The National Academies Press.

Parsazadeh, N., Cheng, P. Y., Wu, T. T., & **Huang, Y. M.** (2021). Integrating computational thinking concept into digital storytelling to improve learners' motivation and performance. *Journal of Educational Computing Research, 59*(3), 470–495. DOI: https://doi.org/10.1177/0735633120967315

Rich, P. J., Jones, B., Belikov, O., Yoshikawa, E., & **Perkins, M.** (2017). Computing and engineering in elementary school: The effect of year-long training on elementary teacher self-efficacy and beliefs about teaching computing and engineering. *International Journal of Computer Science Education in Schools, 1*(1), 1–20. DOI: https://doi.org/10.21585/ijcses.v1i1.6

Rodrigues da Silva, D., Kurtz, F. D., & **Paludo Santos, C.** (2020). Computational thinking and TPACK in science education: a southern-Brazil experience. *Paradigma, 41*(2).

Schanzer, E., Fisler, K., & **Krishnamurthi, S.** (2018). Assessing Bootstrap: Algebra Students on Scaffolded and Unscaffolded Word Problems. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 8–13). ACM. DOI: https://doi.org/10.1145/3159450.3159498

Šiaulys, T., & **Dagienė, V.** (2021, November). Towards classification of interactive non-programming tasks promoting computational thinking. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (pp. 16–28). Cham: Springer. DOI: https://doi.org/10.1007/978-3-030-90228-5_2

Tang, X., Yin, Y., Lin, Q., Hadad, R., & **Zhai, X.** (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education, 148*, 103798. DOI: https://doi.org/10.1016/j.compedu.2019.103798

Tedre, M., Simon, & **Malmi, L.** (2018). Changing aims of computing education: a historical survey. *Computer Science*

Education, 28(2), 158–186. DOI: https://doi.org/10.1080/08993408.2018.1486624

Usta, N., & **Mirasyedioglu, S.** (2021). Conceptual framework for twenty-first century learning: Developing computing and computational thinking. *International Journal of Innovation and Research in Educational Sciences, 8*(5), 280–312.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & **Wilensky, U.** (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127–147. DOI: https://doi.org/10.1007/s10956-015-9581-5

Weston, T. J., Dubow, W. M., & **Kaminsky, A.** (2019). Predicting women's persistence in computer science-and technology-related majors from high school to college. *ACM Transactions on Computing Education*, *20*(1), 1–16. DOI: https://doi.org/10.1145/3343195

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. DOI: https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717–3725. DOI: https://doi.org/10.1098/rsta.2008.0118

Witte, S., Bass, B., O'Byrne, W. I., Price-Dennis, D., & **Sibberson, F.** (2019). Definition of literacy in a digital age. Retrieved 11/11/19, from https://www2.ncte.org/statement/nctes-definition-literacy-digital-age/

Yadav, A., Gretter, S., Good, J., & **McLean, T.** (2017). Computational thinking in teacher education. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 205–220). Cham: Springer. DOI: https://doi.org/10.1007/978-3-319-52691-1_13

Yadav, A., Gretter, S., Hambrusch, S., & **Sands, P.** (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, *26*(4), 235–254. DOI: https://doi.org/10.1080/08993408.2016.1257418

Yadav, A., Lishinski, A., & **Sands, P.** (2021). Self-efficacy profiles for computer science teachers. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (pp. 302–308). DOI: https://doi.org/10.1145/3408877.3432441

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & **Korb, J. T.** (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education, 14*(1), 5–16. DOI: https://doi.org/10.1145/2576872

Yang, H., & **Mouza, C.** (2021). Redesigning educational technology coursework to foster pre-service teacher learning of computational thinking in content area instruction. In C. Mouza, A. Yadav, & A. Ottenbreit-Leftwich (Eds.), *Preparing Pre-Service Teachers to Teach Computer Science* (pp. 129–152). Information Age Publishing, Inc.

TO CITE THIS ARTICLE:
Margulieux, L. E., Enderle, P., Clarke, P. J., King, N., Sullivan, C., Zoss, M., & Many, J. (2022). Integrating Computing into Preservice Teacher Preparation Programs across the Core: Language, Mathematics, and Science. *Journal of Computer Science Integration,* 5(1): 1, pp. 1–16. DOI: https://doi.org/10.26716/jcsi.2022.11.15.35

Submitted: 22 November 2021 **Accepted:** 15 July 2022 **Published:** 15 November 2022