Curriculum for a Comprehensive Statewide In-Service CS Teacher Training Program

Sarah Diesburg
Computer Science
University of Northern Iowa
Cedar Falls, Iowa, USA
sarah.diesburg@uni.edu

J. Ben Schafer
Computer Science
University of Northern Iowa
Cedar Falls, Iowa, USA
ben.schafer@uni.edu

Briana B. Morrison
Computer Science
University of Virginia
Charlottesville, Virginia, USA
bbmorrison@virginia.edu

ABSTRACT

Training teachers to teach high-quality computer science courses is an important step towards increasing participation in computer science. To meet this need, our Research Practitioner Partnership (RPP) was formed with the primary goal of creating a comprehensive, graduate-level program to train in-service CS teachers with little to no prior CS coursework. Since its formation, the RPP has iteratively designed, implemented, and evaluated a five-course program to improve participants' knowledge of computer science content and pedagogy while allowing them to earn the state's grades 5-12 computer science endorsement. Our program has successfully scaled from a single-site pilot to a truly statewide, multi-site program, emphasizing educator-based, standards-based, and cohort-based instruction. Currently, the RPP serves 250 in-service participants.

In this paper, we explain the curriculum development process and delivery model for our program. In doing so, we discuss how the program was intentionally developed from the ground up by designing course offerings that align with the CSTA student and educator standards. We make a case for peer-based Communities of Practice (CoPs) as an essential element of such programs. Finally, we discuss feedback and lessons learned during our ongoing curriculum development process with the hope that these lessons may be valuable for similar organizations looking to create a comparable, comprehensive, CS teacher training program.

CCS CONCEPTS

• **Social and professional topics~Professional topics~Computing education~Computing education programs** •Social and professional topics~Professional topics~Computing education~Model curricula

KEYWORDS

K-12 Education, Teacher Preparation, Curriculum Development, Educator Standards

ACM Reference format:

Sarah Diesburg, Ben Schafer, and Briana Morrison. 2025. Curriculum for a Comprehensive Statewide In-Service CS Teacher Training Program. In *Proceedings of ACM SIGCSE Conference (SIGCSE 2025). ACM, New York, NY, USA, 7 pages.* https://doi.org/10.1145/3641554.3701882

1 Introduction and Background

The University of Northern Iowa Partnership for CS Teacher Preparation is a Research Practitioner Partnership (RPP) [6] founded in 2017 to help in-service teachers become qualified to provide high-quality computer science courses and, in the process, add the grades 5-12 Computer Science endorsement to their teaching license. To meet state requirements, a five-course sequence of graduate level coursework [27] was iteratively created and piloted with the assistance of a 2018 small-sized NSF CS4All:RPP grant (#1738784). Between 2018 and 2021, three cohorts of educators (a total of 49 participants) completed the program using a combination of online instruction and local, face-to-face Communities of Practice (CoP). In 2023, the partnership expanded its membership to a truly statewide program to provide equitable access to course offerings. This effort was supported by a medium-sized NSF CS4All:RPP grant (#2219497) [8]. The partnership is working with two additional cohorts of participants at 9 locations with 94 participants starting the 18-month program in summer of 2023 and 157 starting in summer of 2024.

This paper focuses on the curriculum used in our program. In it we discuss the design principles and delivery model of the program to put the curriculum into context. We discuss our five-course offerings, explaining the structure, learning objectives, and rationale of each. We report mappings to current CSTA student and teacher standards. We offer teacher quotes as evidence of efficacy of the program design as part of our ongoing evaluation. Lastly, we discuss lessons learned with the goal of providing guidance for similar programs.

2 Curricular Design Process

As we launched our RPP, we conducted a literature review for publications discussing similar programs [10, 13, 14, 19, 31]. However, these papers focus on mode of delivery. In this paper, we focus on our curriculum and limit discussions of the delivery model to only those aspects that impact curricular choices.

In designing our curriculum, three guiding principles ground all decisions. Specifically, our program is:

1. ***Educator-based:*** Our audience consists of trained educators and our program is designed with objectives and learning activities appropriate for this audience.
2. ***Standards-based:*** Our program is designed around specific outcomes and standards from the CSTA K-12 Standards for students and the CSTA Standards for CS Teachers.
3. ***Peer/cohort-based:*** Our program is designed with an emphasis on participant collaboration and developing active communities of practice.

We began our design process with the recognition that our program needed to be **educator-based**. Participants are in-service teachers who are well-qualified to teach but, for the most part, lack significant training in computer science. In most cases, the participants have been asked (or told) by their school districts to teach a computer science course to meet the state's K-12 CS requirements. At best, participants have received minimal Professional Development (somewhere between a day and a week) focusing on a specific CS curriculum they will use in their classroom. But frequently, they have received no training at all and were just told to "make it happen." To become well-qualified CS teachers, these participants need to receive a breadth of general knowledge with appropriate depth in particular topics. The focus of that knowledge should be uniquely different from that taught in similar courses to undergraduate compute science majors. Given this audience and focus, our RPP made the decision to build a program from all new courses rather than piecing it together from existing, majors-based courses.

From there, we set out to design our curriculum to be **standards-based**. Using the "Understanding by Design" (UBD) process [3, 16] we began by identifying the learning outcomes and the standards to which those outcomes are connected. Doing so grounds all decisions about course structure and delivery to agreed upon outcomes/standards. We use the CSTA Standards for K-12 students (in particular, grade bands 2 and 3A) [34] and the CSTA Standards for CS Teachers [35]. A deeper discussion of this process is discussed in section 4 of this paper.

Teaching to full-time teachers requires acknowledging and working around participant time-constraints. To address this, we elected to create courses which are, for the most part, asynchronous and online. Participants are provided a schedule of weekly readings, online videos, reflections, and practice problems. They are largely free to complete these activities around their schedule. Some courses include elements of online, small group collaboration such as completing discussion questions or paired programming assignments. The instructional team works with participants to create groups who can identify time(s) to collaborate each week.

Despite these online collaborations, the RPP was concerned about participant feelings of isolation and inadequacy. We decided early on that we wanted to provide participants with opportunities to develop a sense of belonging and combat the very real and significant hurdle presented by Imposter Syndrome [24]. To address this issue, we chose to make our program **peer/cohort-based** through the inclusion of regular Community of Practice (CoP) events. CoPs are defined as a group of people who "share a concern or a passion for something they do and learn how to do it better as they interact regularly."[30] These events help participants not only generate a sense of belonging but also create a peer-based support network for the duration of the program and beyond [18, 29, 30]. Within our program, each of the five courses includes three, half-day, CoP events (approximately monthly) where participants meet face-to-face. These events are well-suited to meet curriculum objectives that aren't easily conducted in an online environment, such as:

- Building community and better understanding peers
- Connecting course content to issues of teaching and learning at various grade bands in the K-12 pathway
- Debating social and ethical issues relating to course content
- Participating in hands-on activities to both better understand course content and to experience these teaching methods
- Practicing teaching techniques
- Interacting to complete group projects
- Collaborating with peers to create shared resources

In early offerings of the program, these CoP events were conducted by university faculty at a single location – either on campus at or at an off-campus site more conveniently located in the state to that cohort's participants. Our current statewide model engages regional partners to conduct these CoP events at multiple sites around the state (Figure 1). University faculty continue as the primary leaders of instruction for classes but are joined on the instructional team by facilitators who lead the face-to-face CoP sessions at one of nine regional partner sites.

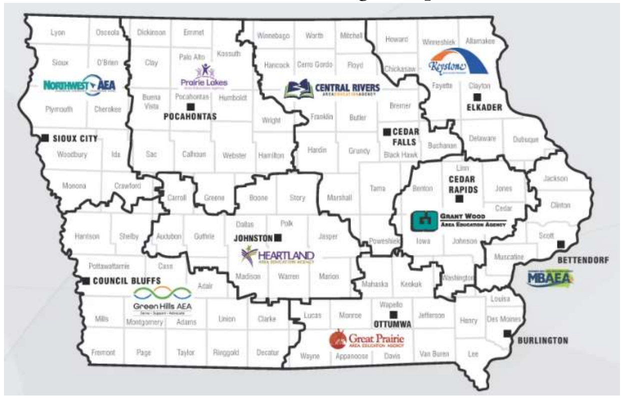

Figure 1: Regional Partnerships facilitating CoP events.

3 Curriculum and Course Outcomes

New cohorts of our program begin each summer, and participants take one course at a time completing the program, and the accompanying endorsement, in December of the following year. In this section, we provide an overview of our program and the learning outcomes for each of the individual courses. Additional information is available at https://csed-uni.github.io.

3.1 Foundational Concepts of CS

FCCS is the first course in the program and is taught as an 8-week summer course. FCCS provides knowledge of the breadth of computing, excluding programming. The textbook used is "Computer Science: An Overview" by Brookshear and Brylow [4], supplemented with faculty-produced instructional videos and a variety of online resources.

Table 1: FCCS Outcomes

Course-Wide Outcomes	
0.1	Analyze and discuss common social and ethical considerations of real-world applications.
Module 1: Data Representation	
1.1	Recognize that the fundamental building blocks of computers are logic gates and, given their inputs, be able to determine the output of a simple collection of gates.
1.2	Apply an understanding of how computers represent various types of values (e.g. bits, bytes, binary, hexadecimal, encodings, storage units).
1.3	Recognize common errors (e.g. overflow and truncation).
Module 2: Hardware, Data Manipulation, and OS	
2.1	Apply understanding of a CPU's instruction set and the instruction cycle to various scenarios.
2.2	Identify hardware components of a computer and describe their relationship and interaction.
2.3	Explain the process whereby a computer's CPU can be connected to or communicate with a variety of external (peripheral) devices.
2.4	Apply an understanding of computer memory/storage to a variety of situations.
2.5	Apply understanding of the role and functioning of operating systems to a variety of situations.
Module 3: Networks and Databases	
3.1	Explain how a network consists of several autonomous systems communicating through established protocols.
3.2	Explain how the Internet consists of multiple networks connected through packet switching.
3.3	Describe how the Web is an example protocol used on the Internet that displays web pages in a client-server model.
3.4	Explain how cybersecurity is an important concern for networks and the software that is built on them.
3.5	Recognize fundamental knowledge of the role, structure, and characteristics of database systems.
Module 4: Artificial Intelligence	
4.1	Differentiate between the concepts of machine reasoning/behavior and human reasoning/behavior.
4.2	Identify common vocabulary concerning artificial intelligence.
4.3	Identify challenges with artificial intelligence concerning images and language processing.

3.2 Fundamentals of Programming

FOP is the second course in the program and is taught as a 16-week fall course. FOP teaches fundamental programming concepts in terms of programming as a discipline, but also considering how teachers and students might use programming as a tool to solve their own problems. The course is taught using Scratch [36] and Python [33], and uses the free online textbook "Python for Everybody" [25] from Runestone Interactive.

Table 2: FOP Outcomes

Course-Wide Outcomes	
0.1	Explain the concepts of sequence, loops, parallelism, events, conditionals, operators, variables, and lists within the context of computer science.
0.2	Given an introductory programming vocabulary term, [write an accurate definition; provide a non-computer example illustrating the vocabulary; explain how the vocabulary is present in a given real-world scenario].
0.3	Consider a block of code and identify its outcome.
0.4	Consider a provided scenario and a block of code that attempts to solve the scenario. Identify whether the code will accurately solve the scenario and, if not, how to fix the code.
0.5	Discuss basic elements of instruction regarding key concepts of computer science in the context of a K-12 classroom.
Module 1: Programming with Scratch	
1.1	Create and debug programs using various forms of interaction, control statements, and functions
Module 2: Beginning Programming with Python	
2.1	Create and debug basic programs using basic data structures, control statements, and functions.
Module 3: Data Analysis with Python	
3.1	Create and debug basic analysis programs using CSV and text-based data files.

3.3 Teaching & Learning of Programming

TLP is the third course in the program and is taught as a 16- week spring course. TLP focuses on pedagogy of programming and supports participants becoming a reflective practitioner [15] through journal entries and small group activities. The textbook is "The Big Book of Computing Pedagogy" by the Raspberry Pi Foundation [7] and course activities and learning outcomes are applied cyclically to the 12 principles of computing pedagogy [23].

Table 3: TLP Outcomes

Course-Wide Outcomes	
0.1	Identify programming fundamentals and discuss prerequisite relationships.
0.2	Analyze programming language considerations for a classroom.
0.3	Explain the program design process.
0.4	Identify aspects of quality code.
0.5	Recognize the presence/absences of quality elements and suggest improvements.
0.6	Discuss teaching/learning beliefs related to programming instruction.
0.7	Identify learning considerations.
0.8	Discuss supportive practices in general and in the context of a specific scenario/classroom.
0.9	Apply programming-based considerations to instructional design.

3.4 Methods of Computer Science

The fourth course in the program is Methods which is taught as an 8-week summer course during the second year This class focuses more broadly on the teaching and learning of computer science rather than only programming. The first half of the class is broken down into exploring and analyzing materials focusing on computer science as a K-12 discipline via standards [1, 34, 37], methods for teaching computer science [2, 5, 9, 12, 17, 20–22, 28, 32, 38–41] , curriculum design [11, 16], and computer science professionalism [15, 42]. The second half of the class is a major group project to produce a course design document for a grade-appropriate course that the participants will teach in their own classrooms.

Table 4: Methods Outcomes

Course-Wide Outcomes	
0.1	Use the national high school computer science standards, analyze potential learning difficulties, and plan teaching for students with different needs.
0.2	For a given course or unit, propose, appropriate student outcomes, assessments, learning activities, and mechanisms for providing feedback and grades to students.
0.3	Describe a variety of instructional outcomes included in middle and high school computer science.
0.4	Describe a variety of methods in the teaching process, including meaningful learning, collaborative learning, inquiry learning, etc. as well as identify the CS instructional outcomes for which each is useful.
0.5	Contribute to a repository of resources for teaching computer science, including materials, lab assignments, class activities, and assessments.

3.5 Data Structures and Algorithms

The final course in the program, DSA is taught as a 16-week fall course. DSA covers topics from algorithms and data structures that teachers may need to support more advanced student projects or teach an AP Computer Science A [43] course. We use "Problem Solving with Data Structures and Algorithms" from Runestone Interactive [26], as well as readings from the textbook previously used in FCCS.

Table 5: DSA Outcomes

Course-Wide Outcomes	
0.1	Apply appropriate terminology when describing the characteristics, advantages, and limitations of different data structures such as array, stack, queue, tree, graph, dictionary, and hash table.
0.2	Research a new data structure or algorithm not previously covered in the course, using credible sources, to understand its purpose and application. Summarize the algorithm's key components for peers/students.
Module 1: Object-Oriented Programming	
1.1	Identify and explain the key concepts of object-oriented programming, including classes, objects, methods, inheritance, polymorphism, encapsulation, and abstraction.
1.2	Recognize and describe the purpose and structure of Python classes and objects in a provided code snippet.
Module 2: Algorithm Analysis	
2.1	Employ appropriate vocabulary to discuss algorithmic efficiency, including terms like Big O notation, time complexity, and space complexity.
2.2	Analyze code to determine its execution-time (big-oh notation) and storage utilization.
Module 3: Linear Data Structures	
3.1	Trace, identify and explain common "linear" data structures constructed using "arrays" (i.e., contiguous block of memory) and "linked nodes" as appropriate: stack, queue, and list.
Module 4: Recursion	
4.1	Define recursion and identify the components of a recursive function, including the base case and the recursive case.
4.2	Trace the execution of a recursive function, demonstrating understanding by outlining the calls and return values step-by-step.
Module 5: Applications	
5.1	Trace, explain, and analyze common search/sort techniques such as linear search, binary search, closed-address hashing.
5.2	Explain and analyze simple and advanced sorts such as bubble, selection, insertion, merge, and quick sorts.

4 Mapping to Standards

Two of the ongoing discussions we have had when developing these course outcomes are 1) why are we requiring this outcome and 2) how does this outcome help prepare better teachers? To guide these discussions, we elected to connect our program and course outcomes to the most appropriate standards faced by computer science teachers in our state – the CSTA K-12 Standards for Students and the CS Standards for CS Teachers.

The CSTA K-12 Computer Science Standards [34] define the skills and key CS knowledge that students should have at various checkpoints along their K-12 journey. The standards are divided into five grade bands. Levels 1A and 1B correspond to elementary school (grades K-2 and 3-5 respectively), level 2 corresponds to middle school (grades 6-8), and levels 3A and 3B correspond to grades 9-12 with level 3A representing a typical student and 3B intended for students pursuing specialty or elective CS courses. Since our program addresses the state's grades 5-12 Computer Science Endorsement, we focused on making sure that our outcomes considered the CSTA standards in levels 2 and 3A.

The CSTA Standards for CS Teachers establish robust benchmarks for teachers who prepare students to meet CS learning outcomes/standards [35]. Specifically, K-12 CS teachers are asked to use these standards to reflect on their own areas of growth, set professional goals, and identify targeted pathways to meet these goals. Unlike the student standards, which explicitly address where students should be at various points in their K-12 career, the teacher standards serve as guideposts or benchmarks that teachers can use to guide their improvement as they gain experience in teaching both in general and in the CS classroom.

4.1 CSTA 2 and 3A

As we considered the overall efficacy and coverage of our program objectives, we began by mapping course objectives and concepts to the K-12 Standards for students. Tables 6 and 7 address the CSTA 2 and CSTA 3A standards, respectively. For each standard, we report whether the standard is "Met," "Progressing" or "Unmet" in our program and, as appropriate, in which course the standards are addressed.

Table 6: Mapping to CSTA Grade Band 2 Standards

Concept	Standard	Status	Class Covered
Computing Systems	2-CS-01,02,03	Unmet	-
Networks & The Internet	2-NI-04,05	Met	FCCS, FOP
	2-NI-06	Progressing	FCCS
Data & Analysis	2-DA-07,08	Met	FCCS, FOP
	2-DA-09	Unmet	-
Algorithms & Programming	2-AP-10,11,12, 13,14,15,16,19	Met	FOP, DSA
	2-AP-17	Progressing	FOP
	2-AP-18	Unmet	-
Impacts of Computing	2-IC-20,21,22,23	Met	FCCS, TLP, DSA

Table 7: Mapping to CSTA Grade Band 3A Standards

Concept	Standard	Status	Class Covered
Computing Systems	3A-CS-01,02	Met	FCCS
	3A-CS-03	Unmet	-
Networks & The Internet	3A-NI-04,05,06,07	Met	FCCS
	3A-NI-08	Progressing	FCCS
Data & Analysis	3A-DA-09,10	Meeting	FCCS
	3A-DA-12	Progressing	FCCS
	3A-DA-11	Unmet	-
Algorithms & Programming	3A-AP-13,14,15,16,17,22	Meeting	FOP, TLP, DSA
	3A-AP-18,19	Progressing	FOP
	3A-AP-20,21,23	Unmet	-
Impacts of Computing	3A-IC-26,27,29,30	Meeting	FCCS, FOP, TLP, Methods
	3A-IC-24,28	Progressing	FCCS, TLP
	3A-IC-25	Unmet	-

As we have iteratively designed and improved our curriculum, we have made continued efforts to understand how our program aligns with these standards. It is important to note that saying this does not mean that we have 100% alignment (as can be seen from the tables above). We find that, at times, we must make thoughtful tradeoffs in balancing two of our program principles: standards-based content versus educator-based content. For example, our Foundations of Programming course (heavily standards-based) is designed to, as much as possible, model the pedagogical techniques that students will learn in TLP (heavily educator-based). Because of this, we must balance the content that we teach and how it is taught.

In the tables above, the term "Progressing" means that we cover some topics related to the standard, but our activities are either evaluated lower in Bloom's Taxonomy than the language employed in the standard, or else, we meet part of, but not all, the full standard. The "Unmet" standards fall into three categories:

Lack of physical computing/devices: Some standards rely on physical systems such as robots. Our program courses are designed to be taken online and asynchronous, which makes incorporating devices challenging.

Lack of data science emphasis: Some standards require large-scale gathering of data, cleaning and visualizing of data sets and encompass data science. While data science concepts are an important part of computer science, our program does not currently emphasize this due to time constraints.

Lack of ongoing software development projects: Some standards include the application of software development models, timeline and collaborative tools, and robust testing. Again, while important, our current program does not emphasize these concepts.

Much like the CSTA Teacher Standards are designed to be benchmarks used to guide teacher improvement, we have selected which Student Standards we feel we can best prepare teachers to meet in our program with its current structure. As the program continues to evolve, we plan to reevaluate these standards and use them as a guide for ongoing program improvement.

4.2 CSTA Teacher Standards

Table 8 shows how the full set of the CSTA Teacher Standards are addressed in our program, whether in one class, multiple classes, or program wide. In addition to addressing these over the lifetime of the program, we engage students in self-reflection using these standards as a significant class activity in the Methods course. Students review each substandard and rate themselves on each of the teacher standards as "not yet", "proficient", or "advanced". Students also self-identify their current experience based on categories defined in the CSTA Roadmap for Professional Learning "Self-Reflection Checklist" [44] and use this as a tool for setting future goals.

Table 8: Mapping to CSTA Standards for CS Teachers

Concept	Standard	Class Covered
CS Knowledge & Skills	1a	TLP
	1b,1c,1f	FCCS
	1d	FOP
	1d,1e	DSA
Equity and Inclusion	2a,2b,2c,	TLP
	2a,2b,2c,2d,2e	Methods
Professional Growth and Identity	3c,3d,3e	TLP
	3a,3d,3e	Methods
	3b,3f	Program-Wide
Instructional Design	4c,4d,4e	TLP
	4a,4b,4c,4d,4f,4g	Methods
Classroom Practice	5b,5c	TLP
	5c	Methods
	5a,5d,5e,5f	Program-Wide

5 Evaluation and Feedback

Formal evaluation of the program is ongoing as we gather feedback from facilitators and participants. We have considerable evidence that most participants find the courses both challenging and rewarding and feel that the program will adequately prepare them to teach computing in their school environment. A complete quantitative analysis of these findings is beyond the scope of this paper. However, we share some evidence that our curriculum is making a positive impact by revisiting our three guiding principles and sharing teacher quotes that support our beliefs that these were meaningful places to start.

Educator-based

- I enjoy the reflection writing and being able to demonstrate and make connections between the reading and my classroom.
- The idea of the "imposter syndrome" and how to help students feel more confident in computer science and all content areas.
- Backwards design is something I'm rolling with heavy!
- Prior to this course, I was able to explain how to do the coding activities, but I couldn't explain why we were doing it that way or why it was important.
- You present the content in many different modes: the textbook, external videos, practice problems, and your videos. We are really focusing on Universal Design for Learning (UDL), and you are providing me with good examples of it.
- I liked having the list of resources on different CS Teaching Methods. It will be great to refer back to this!

Standards-based

- Taking a deep dive into the standards helped clarify how to structure a class.
- Looking closely at the standards to develop outcomes - using UbD backward design - is something that will help me in my course development.
- I think the emphasis on looking at, and considering, standards is always a useful exercise. It makes me reflect on not just CS but also my Social Studies class.
- Taking an inventory of where I am at on teaching the standards. I feel confident that I can continue this self-reflection strategy and update it as I grow in my teaching.
- I am now familiar with CS standards and at least have some knowledge of the goals for computer science education.
- Thinking about course goals for myself, not relying on canned curriculum
- I have really enjoyed looking at the CSTA teacher standards, I have only briefly looked at them in the past so getting some time to unpack them had some ideas going.

Peer/Cohort-based

- I thoroughly enjoy the CoP time and meeting with others who are in the trenches.
- Keep using CoPs. It's really reaffirming when we get together in person to share how the course is going.
- I loved meeting as a group and being able to talk with others and see what they do differently.
- I enjoy listening to the perspectives and input of the other teachers. It lets me know I am not alone.
- Meeting together [at the CoPs] is good. Meeting via Zoom all the time gets tedious.
- The hands-on activities at the CoP are the best part.
- The [online] small groups are great. I know this is hard for some people to make the time to get together, but I REALLY feel this is a valuable part of the process.

Finally, one of the current facilitators wrote the following about teacher participation at a Methods CoP event (the fourth course in the sequence) where small groups were working on their course design documents:

> "I wish they could look back and see themselves through my eyes [to realize] how far they have progressed. It has been outstanding to just hear their conversations."

6 Reflections and Conclusions

Our experiences with the iterative process of curriculum design, implementation, and review have led us to focus on three main lessons learned and worth sharing with others:

(1) Curriculum needs to be designed intentionally, keeping in mind your organization's big principles. We needed to go into our program development knowing our big-picture principles. When we've been faced with challenges, we come back to the idea of how does the decision tie back to these 3 principles. When we have faced challenges in the design process, we often have realized that we had been making decisions without properly remembering our goals. Designing with clear goals/principles at the program's roots grounds the decision-making process.

(2) Balancing face-to-face with asynchronous teaching is important. Based on feedback and evaluation, we continually receive comments that the CoPs help students learn material and gain confidence. We believe that even small amounts of face-to-face meetings can vastly improve the implementation and understanding of certain types of curricula ***as well as*** build community in a way that cannot be done purely online.

(3) As course adjustments are made, the big principles need to be remembered. Being a reflective teacher means that, as courses are taught, adjustments are made to better serve participants. As content or delivery methods change based on lessons learned and participant feedback, it becomes helpful, even necessary, to build in time to revisit program principles, and as necessary, objectives/standards documents, and confirm that changes are in line with program outcomes.

In conclusion, training teachers to teach high-quality computer science courses is challenging. But this paper demonstrates that intentional design of such programs, focusing on meaningful content (objectives) and delivery methods, as well as on supporting participant needs through Communities of Practice, can produce a high impact program. It is our hope that sharing our model and lessons learned proves valuable for organizations looking to create a comparable, comprehensive, CS teacher training program.

ACKNOWLEDGMENTS

This work was funded by two NSF grants (1738784 and 2219497). Thanks to Erin Chute, Philip East, Michelle Friend Michelle Meier, and Corey Rogers for their assistance with our program and ongoing collaboration as RPP partners.

REFERENCES

[1] Lisa Albers. Iowa Computer Science Standards - June 15, 2019.

[2] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An Analysis of Using Many Small Programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019. ACM, 585–591.

[3] Ryan Bowen. Understanding by Design. *Vanderbilt University Center for Teaching*. Retrieved June 18, 2024 from https://cft.vanderbilt.edu/guides-sub-pages/understanding-by-design/

[4] J. Glenn Brookshear and Dennis Brylow. 2020. *Computer science: an overview (13th Edition)*. Pearson.

[5] Adam S. Carter and Christopher D. Hundhausen. 2011. A review of studio-based learning in computer science. *Journal of Computing Sciences in Colleges* 27, 1 (2011), 105–111.

[6] Cynthia E. Coburn, William R. Penuel, and Kimberly E. Geil. 2013. Research-practice partnerships: A strategy for leveraging research for educational improvement in school districts. *William T. Grant Foundation* (2013). Retrieved July 10, 2024 from https://eric.ed.gov/?id=ED568396

[7] Gemma Coleman. 2021. *The big book of computing pedagogy*. Raspberry Pi Foundation, Cambridge, UK. Retrieved from https://www.raspberrypi.org/hello-world/issues/the-big-book-of-computing-pedagogy

[8] Sarah M. Diesburg and J. Ben Schafer. 2023. Scaling Up to a Statewide Network for CS Teacher Preparation by Introducing AEA Community of Practice Partnerships. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 2, SIGCSE 2023, Toronto, ON, Canada, March 15-18, 2023.*, 2023. 1397. https://doi.org/10.1145/3545947.3576341

[9] Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. 2017. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, November 16, 2017. ACM, Koli Finland, 20–29. https://doi.org/10.1145/3141880.3141895

[10] Cameron L. Fadjo, Ted Brown, and Leigh Ann DeLyser. 2013. A curriculum model for preparing K-12 computer science teachers. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, 2013. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[11] Dan Garcia, Maria Camarena, Kevin Lin, and Jill Westerland. 2023. Equitable Grading Best Practices. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2* (*SIGCSE 2023*), March 06, 2023. Association for Computing Machinery, New York, NY, USA, 1200–1201. https://doi.org/10.1145/3545947.3569602

[12] Mark Guzdial. 2003. A media computation course for non-majors. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, 2003. 104–108. Retrieved June 20, 2024 from https://dl.acm.org/doi/abs/10.1145/961511.961542

[13] Helen H. Hu, Cecily Heiner, Thomas Gagne, and Carl Lyman. 2017. Building a Statewide Computer Science Teacher Pipeline. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, March 08, 2017. ACM, Seattle Washington USA, 291–296. https://doi.org/10.1145/3017680.3017788

[14] Alark Joshi, Amit Jain, Ernie Covelli, Jyh-haw Yeh, and Tim Andersen. 2019. A sustainable model for high-school teacher preparation in computer science. In *2019 IEEE Frontiers in Education Conference (FIE)*, 2019.

[15] David A. Kolb. 2014. *Experiential learning: Experience as the source of learning and development.* FT press.

[16] Jay McTighe and Grant Wiggins. 2012. Understanding by design framework. *Alexandria, VA: Association for Supervision and Curriculum Development* (2012). Retrieved June 20, 2024 from https://www.sabes.org/sites/default/files/news/5_UbD_WhitePaper0312%5B1%5D.pdf

[17] Alvaro E. Monge, Cameron L. Fadjo, Beth A. Quinn, and Lecia J. Barker. 2015. EngageCSEdu: Engaging and Retaining CS1 and CS2 Students. *ACM Inroads* 6, 1 (February 2015), 6–11. https://doi.org/10.1145/2714569

[18] Lijun Ni, Mark Guzdial, Allison Elliott Tew, Briana Morrison, and Ria Galanos. 2011. Building a community to support HS CS teachers: the disciplinary commons for computing educators. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, March 09, 2011. ACM, Dallas TX USA, 553–558. https://doi.org/10.1145/1953163.1953319

[19] Gamze Ozogul, Mike Karlin, and Anne Ottenbreit-Leftwich. 2018. Preservice teacher computer science preparation: A case study of an undergraduate computer education licensure program. *Journal of Technology and Teacher Education* 26, 3 (2018), 375–409.

[20] Leo Porter, Cynthia Bailey Lee, Beth Simon, Quintin Cutts, and Daniel Zingaro. 2011. Experience report: a multi-classroom report on the value of peer instruction. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, June 27, 2011. ACM, Darmstadt Germany, 138–142. https://doi.org/10.1145/1999747.1999788

[21] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. 2016. A Multi-institutional Study of Peer Instruction in Introductory Computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, February 17, 2016. ACM, Memphis Tennessee USA, 358–363. https://doi.org/10.1145/2839509.2844642

[22] Rathika Rajaravivarma. 2005. A games-based approach for teaching the introductory programming course. *SIGCSE Bull.* 37, 4 (December 2005), 98–102. https://doi.org/10.1145/1113847.1113886

[23] James Robinson. 2020. How we teach computing. *National Centre for Computing Education.* Retrieved June 19, 2024 from https://blog.teachcomputing.org/how-we-teach-computing/

[24] Adam Rosenstein, Aishma Raghu, and Leo Porter. 2020. Identifying the Prevalence of the Impostor Phenomenon Among Computer Science Students. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, February 26, 2020. ACM, Portland OR USA, 30–36. https://doi.org/10.1145/3328778.3366815

[25] Runestone Academy. *Python for Everybody - Interactive.*

[26] Runestone Academy. *Problem Solving with Algorithms and Data Structures 3rd edition.* Retrieved June 20, 2024 from https://runestone.academy/ns/books/published/pythonds3/index.html?mode=browsing

[27] J. Ben Schafer and J. Philip East. 2022. Creating a High Quality, High Impact CS Teacher Prep Program. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, February 22, 2022. ACM, Providence RI USA, 599–605. https://doi.org/10.1145/3478431.3499338

[28] Kimberly Scott, Kimberly M. Sheridan, and Kevin Clark. 2015. Culturally responsive computing. *Learning, Media and Technology* 40, 4 (2015), 412–436.

[29] Josh Tenenberg and Sally Fincher. 2007. Opening the door of the computer science classroom: the *disciplinary commons. SIGCSE Bull.* 39, 1 (March 2007), 514–518. https://doi.org/10.1145/1227504.1227484

[30] Etienne Wenger. 1999. *Communities of practice: Learning, meaning, and identity.* Cambridge university press.

[31] Aman Yadav, Cornelia Connolly, Marc Berges, Christos Chytas, Crystal Franklin, Raquel Hijón-Neira, Victoria Macann, Lauren Margulieux, Anne Ottenbreit-Leftwich, and Jayce R. Warner. 2022. A Review of International Models of Computer Science Teacher Education. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, December 27, 2022. ACM, Dublin Ireland, 65–93. https://doi.org/10.1145/3571785.3574123

[32] 2015. Pair Programming: Does It Really Work? | Agile Alliance. Retrieved June 20, 2024 from https://www.agilealliance.org/glossary/pair-programming/

[33] 2024. Welcome to Python.org. *Python.org.* Retrieved June 18, 2024 from https://www.python.org/

[34] CSTA K–12 Standards. Retrieved June 18, 2024 from https://csteachers.org/k12standards/

[35] CSTA Standards for CS Teachers. Retrieved June 18, 2024 from https://csteachers.org/teacherstandards/

[36] Scratch - Imagine, Program, Share. Retrieved June 18, 2024 from https://scratch.mit.edu/

[37] K–12 Computer Science Framework. *k12cs.org.* Retrieved June 20, 2024 from http://k12cs.org

[38] Teaching Methods. Retrieved June 20, 2024 from https://teach.com/what/teachers-know/teaching-methods/

[39] POGIL | Home. Retrieved June 20, 2024 from https://pogil.org/

[40] CS-POGIL | Home. *CS-POGIL.* Retrieved June 20, 2024 from https://cspogil.org/Home

[41] How Can You Engage A Diverse Range of Girls in Technology? | National Center for Women & Information Technology. Retrieved June 20, 2024 from https://ncwit.org/resource/how-can-you-engage-diverse-range-girls-computing/

[42] Guidance for Reflective Teachers. *Computer Science Teachers Association.* Retrieved June 20, 2024 from https://csteachers.org/reflective-teachers/

[43] AP Computer Science A Standards Mapping for AP Computer Science Principles* | CodeHS. Retrieved June 20, 2024 from https://codehs.com/standards/framework/AP_CSA/course/400

[44] Roadmap for Professional Learning. *Computer Science Teachers Association.* Retrieved June 26, 2024 from https://csteachers.org/roadmap-for-professional-learning/