Programming-Integrated Mathematics Learning for Future Elementary Teachers and Non-STEM Majors

Hyejin Park
hyejin.park@drake.edu
Drake University
Des Moines, Iowa, USA

Eric D. Manley
eric.manley@drake.edu
Drake University
Des Moines, Iowa, USA

ABSTRACT

With the rising emphasis on computer science (CS) education, many states, school districts, and schools have grappled with how to address computing in the K-12 curriculum. While some approaches dedicate classroom time to specific CS topics, others are attempting to infuse computing curriculum into other subjects like math and science. Preparing educators without prior CS experience to teach this integrated content is challenging. In this paper, we share *programming-integrated mathematics learning* (PML) modules that we developed to support teachers in learning to use programming in mathematics classrooms through argumentation. To test the effectiveness of the modules, we implemented them in a mathematics content course for elementary education majors and a mathematics pathway course for non-STEM majors. We assessed both courses as study contexts for evaluating potential PML modules to enhance teachers' knowledge and students' learning outcomes in mathematics and computer programming. This paper focuses on student-written reflections as data to investigate how education and non-STEM majors describe their learning through the PML modules. According to our analysis, the study participants found the PML modules to be useful in enhancing their understanding of mathematical concepts. They also described the value of learning programming in mathematics classrooms through argumentation, learning different perspectives, and helping each other understand. However, most of our student participants displayed a relatively low level of confidence in their programming abilities, specifically regarding the creation of text-based programs. Future study is needed to explore how to increase the programming self-efficacy while students learn mathematics through programming.

CCS CONCEPTS

• **Applied computing** → **Collaborative learning**; **Interactive learning environments**; • **Social and professional topics** → **Computing education**; • **General and reference** → *Design*; • **Human-centered computing** → *Empirical studies in collaborative and social computing*.

KEYWORDS

Mathematics and Programming Integration, Teacher Education, Collaborative Programming, Collective Argumentation

ACM Reference Format:
Hyejin Park and Eric D. Manley. 2024. Programming-Integrated Mathematics Learning for Future Elementary Teachers and Non-STEM Majors. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630908

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03…$15.00
https://doi.org/10.1145/3626252.3630908

1 INTRODUCTION

According to a 2022 State of Computer Science Education report, 43 states are in the process of developing or adopting computer science (CS) standards for K-12 classrooms [8]. This momentum underscores a collective endeavor to diversify and fortify STEM career pipelines. However, many schools find it taxing to allocate dedicated CS resources or expertise. This is exacerbated in many rural areas, which typically lag behind in access to CS and technology education [16]. Thus, the integration of CS into other STEM subjects, such as mathematics or science, is a popular solution but still brings its own challenges. Many elementary teachers themselves have not been exposed to CS much in their teacher education programs [22], and professional learning opportunities to learn CS that teachers receive are mostly sporadic [20]. This, combined with potential apprehensions about their own math or science proficiency, accentuates the challenges. Thus, teacher educators, researchers, and stakeholders need to consider a better way to support elementary teachers to help them build robust CS content knowledge and skills required for CS-integrated teaching. The challenges or barriers teachers encounter in planning and implementing CS-integrated classroom lessons are unsurprising outcomes (e.g., [27]).

As teacher educators shaping the next generation of teachers, it is imperative for us to ensure that future teachers are equipped to navigate this interdisciplinary terrain. In order to help address this need, we developed a series of *programming-integrated mathematics learning* (PML) modules intended to simultaneously address fundamental concepts in CS and mathematics so teachers can experience CS-and-mathematics-integrated learning. Our goal for this paper is to assess the effectiveness of the PML modules through their impact on the learning of prospective teachers. To achieve this objective, we implemented the modules in a mathematics content course designed for elementary education majors. Moreover, we also implemented a subset of the activities in a math pathways course – a general mathematics course for non-STEM majors. The Math Pathways course provides a comprehensive foundation in

mathematics for liberal arts and business students. It fulfills university requirements and prepares students for college-level mathematics. Because this group consists of students with low levels of math proficiency and little programming experience, if their response was positive, we believed the modules could also be used in secondary or post-secondary mathematics classrooms (e.g., developmental mathematics courses). The subsequent sections will review related works, outline questions guiding our examinations of the effectiveness of the modules, discuss our methods for the module evaluations and results from our analysis, and conclude with potential implications and directions for future exploration.

2 RELATED LITERATURE AND CONCEPTUAL FRAMING FOR MODULE DESIGN

Programming is a core component of CS practice, and it has already been integrated into mathematics classrooms and mathematics curricula across many countries to foster students' problem-solving skills and logical thinking [3]. Prior studies examining the benefits of incorporating programming into mathematics classrooms show that programming can help increase students' interest in mathematics, enhance their mathematical performance and reasoning skills, and help students connect mathematics to real-world situations [1]. For instance, Rodríguez-Martínez et al. (2020) performed a controlled study in which two groups of 6th-grade students were trained in basic Scratch (block-based programming) and then given a mathematical task (word problems containing computation of least common multiples and greatest common divisors) [28]. One group used Scratch to complete the mathematical task, and the other used traditional pencil-and-paper approaches. Proficiency in the mathematical task was higher among the Scratch group, leading to the conclusion that it is reasonable to integrate mathematical and computational curricula.

According to Falkner et al.. unplugged, block-based, and text-based programming is being used at all age levels by K-12 schoolteachers [11]. For ages 3-12, block-based programming is more common than text-based programming, and the split is about equal from ages 13-19. Teachers participating in Falkner et al.'s study described various factors that influenced the selection of the programming environment, with curricular demand being fairly low, developmental appropriateness being high, and teacher confidence somewhere in the middle. Other studies show that since most elementary teachers have no or little CS learning and teaching experience, they lack CS content knowledge and do not feel confident in integrating programming into mathematics classrooms [30, 31]. Although professional learning courses or workshops have been offered to teachers to support their learning of CS, teachers appealed that they need more practice, support, and resources that can help them integrate CS into other subject classroom. We believe that providing CS learning opportunities during mathematics education coursework in a teacher education program can help teachers be equipped with both CS and mathematics content knowledge for teaching before they go out to the field. Pointing out that only a few teacher education programs are preparing future CS teachers, DeLyser et al. recommended that schools of education include CS as a part of teacher preparation programs [9]. Gadanidis et al.'s 2017 study shows that student apprehension about CS-math integration was initially high among pre-service teachers but improved with exposure to integrated activities and online resources. They also found that many pre-service teachers had successfully implemented these ideas in a practicum setting [13]. So, in this study, we designed PML modules to have future teachers experience learning mathematics through programming as learners during their mathematics education coursework, hoping to increase their CS content knowledge and skills and their confidence in the ability to use programming in doing mathematics and further in teaching mathematics in their future classrooms. Watson defined self-efficacy to be "confidence or belief in one's own abilities to perform an action or activity necessary to achieve a goal or task" [32] (p. 152). According to prior studies, even short-term professional learning opportunities can improve teachers' confidence in programming [5].

In the PML modules, we used both block-based and text-based programming. We introduced block-based programming first since studies show that students with block-based experience performed better when learning text-based programming [14]. However, when selecting block-based programming platforms, different platform characteristics may or may not support students' transitioning to text-based programming [19]. Many studies also show that teachers have widely used student collaboration to support students' learning of programming. In collaborative programming, students learn from each other by sharing ideas, increasing students' academic success [4, 12]. The discourse students engage in while co-programming can support students' learning better than working alone. In our PML modules, most programming activities involved co-programming. The key feature of our PML module is that we incorporate argumentation to facilitate students' collaborative programming and discourse in solving PML tasks. Argumentation is a core mathematical practice that students are expected to learn in school mathematics [18, 21, 24]. Previous studies have proved that argumentation helps students construct a conceptual understanding of mathematical concepts [29], increase their participation and engagement [7], develop their mathematical disposition and intellectual autonomy [33], and enhance their communication skills [2]. Students tend to make programs using a trial-and-error method without deliberating why programs work [10]. We believe engaging students in argumentation while programming could guide them to focus more on their reasoning and thinking, not relying much on trials. Trial-and-error methods may help to make a conjecture, but then students should be able to prove why something is true or valid for their understanding. This is a crucial practice students should learn both in mathematics and programming.

The following questions guided our study to examine the effectiveness of the PML modules on student learning:
Q1: How do education and non-STEM majors describe their learning mathematics and programming through the PML modules?
Q2: What conceptions do education and non-STEM majors possess about *programming* in learning mathematics through PML modules?
Q3: What conceptions do education and non-STEM majors possess about *collective argumentation* in learning mathematics through PML modules?
Q4: How do education and non-STEM majors describe their confidence level concerning programming after completing the PML modules?

Q5: What are education and non-STEM majors' programming preferences between block-based and text-based programming after they complete the PML modules? Why do they prefer to use one over the other in learning mathematics?

3 LEARNING MODULE DEVELOPMENT AND IMPLEMENTATION

This section describes the development and implementation of our PML modules and the data we used to examine their effectiveness.

3.1 PML Modules and Study Contexts

In the 2023 spring semester, we designed a series of PML modules focusing on argumentation to support teacher knowledge development about mathematics and programming. The mathematics content concentrates on fundamental topics in geometry and probability addressed in school mathematics. To assist teachers' programming learning, we created PML tasks and activities using both block-based and text-based programming. During these activities, students were asked to analyze and interpret programs and then design or revise existing programs to create new ones. The activities were devised this way with the consideration that participants may lack prior programming experience. More specifically, activities with block-based programming (e.g., EdScratch and Scratch) had learners explore characteristics of geometrical shapes (e.g., triangles) and program physical and virtual robots to draw shapes. Each learner had one physical robot (Edison robot) to program and run during in-class activities. We utilized Park et al.'s [25] four geometry learning tasks that combine programming and robotics, designed for teachers to use in mathematics classrooms for all grade levels. We also created a programming-based geometry learning task using Scratch to encourage students to explore further with block-based programming. During the activities, students created programs using both EdScratch and Scratch, and they compared the similarities and differences between the two languages. Activities with text-based programming allowed learners to explore experimental probabilities using programming and to connect experimental and theoretical probabilities. We also designed these PML modules to be centering around argumentation to have students focus on reasoning in programming and problem-solving rather than simply relying on the trial-and-error method to figure out problems.

Our modules consist of in-class group-work-based PML tasks and activities, individual and group assignments created to assess knowledge development about mathematics and programming, and written reflection assignments asking learners to reflect on their learning through the PML modules (see Table 1 for the snapshot of the modules). We devised mathematics and programming content knowledge assessments and written reflection prompts after learners completed in-class group work activities. We did so to consider their reactions to our PML modules in our learning module design and development processes. In all, programming concepts covered included variables, operations, conditionals, loops, and functions.

We implemented the PML modules in two different learning contexts to explore perceptions of learning mathematics through programming from two different groups of populations. To assess the impact of the modules on future teachers, we implemented the modules in a mathematics content course for elementary education majors. Moreover, to further evaluate the learning modules' effectiveness in settings with novice math/CS learners, we also implemented a subset of the activities in a math pathways course – a general mathematics course for non-STEM majors. The first author of this paper was the instructor of these two courses. She taught one section of the mathematics content course and two sections of the math pathways course. During our weekly research meetings throughout the semester, we discussed students' reactions to the PML tasks and activities in class. We then created and revised mathematics and programming content knowledge assessments and written reflection prompts considering students' mathematics and programming knowledge levels and our research interests.

3.2 Student Backgrounds

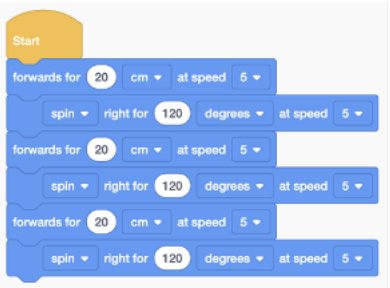
We analyzed written reflections that were submitted by participants of our study. The participants were a group of four elementary education majors who were enrolled in a mathematics content course designed for K-8 teachers (pseudonyms: Blake, Jenna, Kerry, and Stacy), and seven non-STEM majors who were taking a mathematics pathways course (pseudonyms: Aiden, Casey, Riley, Emery, Idris, Jordan, Liam). We selected these participants because they had completed all the reflections that we had requested them to submit. While taking the pathways course, Jordan was a freshman. Casey, Riley, Aiden, Liam were seniors. All education majors were sophomores. These students voluntarily participated in the study and completed all four written reflection assignments. Among these student participants, eight reported they had no programming experience before taking the course. Jenna and Jordan remembered some programming activities they had participated in when they were either middle or high school students. Aiden was the only one who expressed familiarity with programming.

3.3 Analysis of Student Reflections

When interpreting students written responses, we drew on a situative perspective [26] to help us understand their responses in the contexts they were situated in. For instance, education majors in our study were asked to situate themselves as elementary teachers and describe their views on the use of programming in mathematics classrooms focusing on argumentation as teachers. We also asked them to reflect on their learning through the PML module as learners. So, when interpreting their responses, we were careful whether they elaborated their views from a learner's or a teacher's view.

To code student-written reflections, we used thematic analysis [6]. We coded each reflection, paying attention to the parts where each participant provided their views on programming and collective argumentation, their rates on their confidence levels regarding programming, their comments on the PML modules, and their programming preferences in learning mathematics. We began coding education majors' responses from Module 1 Reflection 1, then moved to coding their responses from Module 1 Reflection 2, Module 2 Reflection 1, and Module 2 Reflection 2. We carefully read student responses and created codes using the constant comparison technique. When coding non-STEM majors' responses, we used the codes we came up with from the analysis of education majors'

	Module 1	Module 2

In-class activities	• Introduction and discussion about mathematical argumentation
	• Investigations of definitions, properties, and examples of polygons
	• Mini-lesson on robotics and programming
	• PML activities: Triangle explorations and constructions using block-based programming (EdScratch) and Edison robots.
	Example Activity: The following program is for Edison to draw a shape:
	 
	What shape do you think the program is going to tell Edison to draw? Explain your thinking.
	Using a ruler and protractor, draw the shape Edison would make on the provided paper.
	Now, let's program and run your Edison. Make a program like the program above in EdScratch (https://www.edscratchapp.com/) and program your Edison to see what shape Edison is drawing. Is it moving as you expected?
		• Introduction to Probability
		• PML activities: Explorations of experimental probabilities using text-based programming (Python); Explorations of a relationship between experimental and theoretical probabilities
		Example Activity: Examine each line of code in the program and write an argument in the given box below that describes your interpretations of the program. How did you interpret each line of code?
		`from random import randint`
		
		`count_trials = 1`
		`count_primes = 0`
		`num_trials = 10000`
		
		`while count_trials <= num_trials:`
		`  die1 = randint(1, 6)`
		`  die2 = randint(1, 6)`
		`  total = die1 + die2`
		
		`  if total in [2, 3, 5, 7, 11]:`
		`  count_primes = count_primes + 1`
		
		`  count_trials = count_trials + 1`
		
		`print("Number of primes:", count_primes)`
		`print("Estimated probability:", (count_primes / num_trials))`
		
		Now run the program and see whether it is running as you expected.
		Run the program more than 10 times. What did you notice when you ran the program? Write down whether your interpretations were correct based on your observations.

Individual Assignment	Analysis and evaluation of an already designed EdScratch block-based program to determine why or why not this program would work for drawing a kite	Revision of a text-based program used in class to perform another experiment with dice to find the experimental probability of getting an even number as the sum when throwing two dice simultaneously.
Reflection 1	*Both Education and Non-STEM Majors:* Written reflections about their geometry learning through the PML activities using block-based programming and their views of programming and proving *Education Majors Only:* Written reflections about their views of learning and teaching mathematics using programming, their confidence level in teaching block-based programming, additional support they need to use programming in their future mathematics classrooms	*Both Education and Non-STEM Majors:* Written reflections about their learning experience with text-based programming during their school education, their probability learning through the PML activities using text-based programming, their comfort and confidence level of using block-based and text-based programming to design programs, and their views on the relationship between programming and proving *Education Majors Only:* Written reflections about their views of learning and teaching mathematics using programming, their confidence level in teaching text-based programming, their preference between block-based and text-based programming
Group Assignment	Hexagon explorations and constructions using Block-based programming (Scratch) and virtual robots.	Analysis and evaluation of a text-based program to determine what problem a programmer attempted to solve by running the program.
Reflection 2	Written reflections about their learning experience through argumentation	Written reflections about their collaborative programming experience through argumentation, and their views on the benefits of learning mathematics and programming through *collective argumentation*

Table 1: PML Modules Implemented in Education and Math Pathways Courses

written responses, opening up new codes from their responses. After coding all participants' written reflections, we collated all the codes, identified themes that emerged from participants' responses, and wrote memos about how each group of students, education and non-STEM majors, described their learning through the PML modules and their views on programming and learning mathematics through programming focusing on argumentation.

4 RESULTS

In this section, we illustrate the results of our analysis by answering each question to discuss the effectiveness of the PML modules.

4.1 Q1: Reinforcement of Math Concepts

Most students evaluated that the PML activities helped develop their understanding of mathematical concepts. For instance, ten of the eleven students mentioned that block-based programming activities, the first PML module they were engaged in at the beginning of the semester, positively affected their learning of the geometry concepts. Kerry discussed how she had previously learned the properties of triangles, but that by programming the robot (virtually or physically) during PML activities, she found out why it worked, increasing her understanding of triangle shapes and allowing herself to develop better warrants for her claims (Module 1 Reflection 1). Some of the reasons for this improved mathematical understanding included that programming offers visualization in its process and provides a mechanism for trial and error. Casey notes that "It was nice to be able to see what happens when you change the angles and the sides of different shapes…I learned a lot from these robots also because I am a visual learner " (Module 1 Reflection 1) and Kerry further elaborated that programming "allows for trial and error, which in turn explains why something may/may not work" (Module 1 Reflections 1). However, one student conceded that the block-based programming activities improved her understanding but may not have been optimal because of struggles in learning programming itself (Stacy, Module 1 Reflections 1).

Similarly, students largely (9 of 11) thought that the inclusion of text-based programming activities, which were the second part of the PML module, helped their understanding of probabilities. The theme of visualization continued in these responses, with four students saying something about it, which is interesting because these text-based programming activities were much less visual. Casey noted that "Text-based programming helps me with math because I am able to visualize what I'm thinking in my head which is very helpful" (Module 2 Reflection 1), and Aiden mentioned that "It allowed me to visualize an input and an output" (Module 2 Reflection 1). Students saw that the text-based program provides a means of "visually" expressing an idea and that the program output served as examples reinforcing the content. Emery and Casey commented on the utility of being able to quickly test hypotheses (their claims): "It saved a lot of time and made finding answers a lot simpler" (Emery, Module 2 Reflection 1) and "I was able to calculate things on a larger scale than I could just do in my own head." (Casey, Module 2 Reflection 1). Kerry also discussed that solving the programming tasks required her to think through the details of the mathematics much more deeply in order to get the desired output in the program.

4.2 Q2: Conceptions of Programming

Our study participants had similar conceptions about programming. All students, except Liam, described *programming* as a set of instructions to a computer or a robot to achieve the goal of a task. For instance, Casey explained, "Programming is when you type in problems and try to get an answer out of it. It's like you are giving computer instructions to see what it will do with it" (Module 1 Reflections 1). Unlike other students, when asked to define what programming is, Liam instead focused on types of reasoning or mathematics knowledge needed for programming.

In describing their views on programming, nine students mentioned that proving and programming were similar since both require problem-solving, validation, and justification. The other two students had different thoughts; e.g., Jordan thought that proving and programming were not similar because "proving processes is used to see if something is true through proofs or not while [the] programming process is used to see if something is true through trials" (Jordan, Module 2 Reflection 1). For Jordan, programming needs trial-and-error for verification; proving does not.

When asked to think about the use of programming in their future classrooms (we asked this question only to education majors), from a teacher's perspective, Kelly saw programming as an *assessment tool* that can support teachers in evaluating student understanding of mathematical concepts (Module 1 Reflections 1). The other three students perceived programming as a *learning tool*, addressing that programming could be used to help students develop mathematical understanding. Jenna commented, "I think it is [a] helpful way to further students' understanding of mathematical terms because students will be able to type out their own mathematical program and then watch their program play out and if it works or doesn't work" (Module 2 Reflection 1).

4.3 Q3: Conceptions of Collective Argumentation

Most students (9 out of 11) defined *collective argumentation* as the process of a group of people making arguments. Some students specified that it is a process of convincing themselves or others about why something is correct. For example, Idris defined collective argumentation in the following way: "I would define collective argumentation as the process of a group of people working together on a problem and sharing their thoughts and working before coming to a group consensus on what the answer is" (Module 2 Reflection 1). When asked to describe their thoughts on the benefits of learning mathematics and programming through collective argumentation, based on their experience working with peers/groups through argumentation during the PML activities, many students responded that in argumentation, they could share ideas so they can learn different perspectives or thinking, and they also help each other to understand. Blake stated, "The benefits of learning mathematics and programming through collective argumentation are that if we don't understand something about how to solve a problem or how to provide proof, then we can ask our partners how they understand the problems for other perspectives and reasonings" (Module 2 Reflection 2). One student, Liam, added other benefits, addressing that argumentation can help enhance communication

skills and gain a deeper understanding of concepts discussed in class.

4.4 Q4: Confidence Level

After completing the PML activities at the end of the semester, students were asked to rate their confidence in their ability to create text-based programs on a scale of 0-3, representing (0) not at all confident, (1) minimally confident, (2) quite confident, and (3) I can do it effectively. Five students rated themselves as 1, four rated themselves a 2, and the other two rated themselves between a 1 and a 2. As expected with beginning programmers, the students felt much more confident working with sample code and less so creating an entirely new program by themselves. Kerry summarizes this feeling by saying, "I am pretty good at manipulating existing programs to reflect new sequences or probabilities but do struggle on creating them from scratch" (Module 2 Reflection 1).

When the education majors were asked to elaborate on their confidence in teaching block-based and text-based programming, they were much more confident in their ability to teach block-based. They also mentioned they were more confident making or using block-based programs than text-based ones. For teaching block-based programming, with the same scale of 0-3 described above, one student rated themselves a 1, and the other three rated themselves as a 2. For teaching text-based programming, one student rated themselves a 1, one rated themselves a 2, and the others were unsure. For teaching purposes, the education majors felt the need for more practice with programming as well as resources with pre-written example programs for teaching mathematics through programming. When reflecting on the support she would need to teach programming in her future mathematics classroom, Jenna said, "I just believe that overall, I need more practice. More practice with the designing a code and revising if it goes wrong" (Module 2 Reflection 1). And Kerry noted, "I like that if I had pre-existing programs, I could use those to support my students with their learning about text-based programming, but I am not sure if I have the proper knowledge needed to explain how to create the programming or how it works in computing the results" (Module 2 Reflection 1).

4.5 Q5: Programming Preferences

When asked about their preferences between block-based and text-based programming, we noticed a difference between the education majors and students in the general math course. The general math group (non-STEM majors) was evenly split (with 3 preferring block-based, 3 preferring text-based, and 1 with no preference). In contrast, all education majors preferred block-based programming. Each of the education majors reflected on this through the lens of what they might use in a future mathematics classroom, and since they perceived block-based programming as more appropriate in an elementary education setting, that was their preference. Blake said, "I found that Scratch [block-based programming] was a lot more simple to use, and I figured that out very fast and felt [elementary] students would be able to. It is very interactive and has lots of different options as well as a cat to draw out what you are doing, which is a fun interactive tool for them" (Module 2 Reflection 1). This idea of being able to select from different options also appealed to the general math students. Aiden noted, "I think block-based could be easier because there are options, and text-based, I feel like I just make something up" (Module 2 Reflection 1). However, some of the general math students found the text-based Python code easier to interpret: "I think I would prefer text-based simply for the fact that it is spelled out for you to understand. With block programming, sometimes you had to guess what it was used for or what it's purpose was" (Emery, Module 2 Reflection 1).

5 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we shared our experiences through the analysis of student-written reflections. Overall, both groups of students, education and non-STEM majors, evaluated that the PML modules helped them improve their understanding of mathematical concepts. They also constructed similar views on the benefits of using programming and argumentation in learning mathematics. We also observed that the PML modules fostered education majors to increase self-efficacy in teaching block-based programming. The programming choice that they wanted to use in teaching mathematics was block-based, which aligns with their confidence levels in their ability to use and teach block-based programming. In contrast, their confidence levels in teaching text-based programming were minimal, aligning with their confidence levels in creating text-based programs. Non-stem majors were also not confident in their ability to produce text-based programs. Our results are similar to Kahle's 2008 study, which shows a strong relationship among teachers' mathematics self-efficacy, mathematics teaching self-efficacy, and teaching method choices [17]. Although our study participant sample size was not big enough to generalize, our results still give us insight into the importance of developing teachers' self-efficacy and teaching self-efficacy about programming since these can affect teachers' teaching practices. When supporting teachers in learning to integrate programming in mathematics classrooms, we need to think about ways to improve teachers' self-efficacy and teaching self-efficacy about programming in addition to their mathematics self-efficacy and mathematics teaching self-efficacy. Research also shows a strong relationship between students' mathematics self-efficacy and mathematical performance (e.g., [15, 23]). Future studies are needed to explore how we can support students and (future) teachers to increase self-efficacy about programming in CS-and-mathematics-integrated learning contexts.

REFERENCES

[1] Louis Alfieri, Ross Higashi, Robin Shoop, and Christian D Schunn. 2015. Case studies of a robot-based game to shape interests and hone proportional reasoning skills. *International Journal of STEM Education* 2, 1 (2015), 1–13.
[2] J ANDRIESSEN. 2006. Arguing to learn. *Cambridge handbook of the learning sciences* (2006), 443–459.
[3] Anja Balanskat and Katja Engelhardt. 2014. *Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe.* European Schoolnet.
[4] Moshe Barak and Muhammad Assal. 2018. Robotics and STEM learning: students' achievements in assignments according to the P3 Task Taxonomy—practice, problem solving, and projects. *International Journal of Technology and Design Education* 28 (2018), 121–144.
[5] Marina Bers, Safoura Seddighin, and Amanda Sullivan. 2013. Ready for robotics: Bringing together the T and E of STEM in early childhood teacher education. *Journal of Technology and Teacher Education* 21, 3 (2013), 355–377.
[6] Virginia Braun and Victoria Clarke. 2012. *Thematic analysis.* American Psychological Association.

[7] Marta Civil and Roberta Hunter. 2015. Participation of non-dominant students in argumentation in the mathematics classroom. *Intercultural Education* 26, 4 (2015), 296–312.

[8] Code.org, CSTA, and ECEP Alliance. 2022. 2022 State of computer science education: Understanding our national imperative. Retrieved from https://advocacy.code.org/stateofcs.

[9] Leigh Ann DeLyser, Joanna Goode, Mark Guzdial, Yasmin Kafai, and Aman Yadav. 2018. Priming the computer science teacher pump: Integrating computer science education into schools of education. *CSforAll, New York, NY* (2018).

[10] Stephen H Edwards. 2004. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. 26–30.

[11] Katrina Falkner, Sue Sentance, Rebecca Vivian, Sarah Barksdale, Leonard Busuttil, Elizabeth Cole, Christine Liebe, Francesco Maiorana, Monica M McGill, and Keith Quille. 2019. An international comparison of k-12 computer science education intended and enacted curricula. In *Proceedings of the 19th Koli calling international conference on computing education research*. 1–10.

[12] Garry Falloon. 2016. An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. *Journal of Computer Assisted Learning* 32, 6 (2016), 576–593.

[13] George Gadanidis, Rosa Cendros, Lisa Floyd, and Immaculate Namukasa. 2017. Computational thinking in mathematics teacher education. *Contemporary Issues in Technology and Teacher Education* 17, 4 (2017), 458–477.

[14] Marcos J Gomez, Marco Moresi, and Luciana Benotti. 2019. Text-based programming in elementary school: a comparative study of programming abilities in children with and without block-based experience. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. 402–408.

[15] Gail Hackett and Nancy E Betz. 1989. An exploration of the mathematics self-efficacy/mathematics performance correspondence. *Journal for research in Mathematics Education* 20, 3 (1989), 261–273.

[16] Google Inc. and Gallup Inc. 2017. Computer Science Learning: Closing the Gap: Rural and Small Town School Districts. Retrieved from https://goo.gl/hYxqCr. *Results From the 2015-2016 Google-Gallup Study of Computer Science in U.S. K-12 Schools (Issue Brief No. 4)* (August 2017).

[17] Diane Kay Kahle. 2008. *How elementary school teachers mathematical self-efficacy and mathematics teaching self-efficacy relate to conceptually and procedurally oriented teaching practices*. Ph. D. Dissertation. The Ohio State University.

[18] Steve Leinwand. 2014. *Principles to actions: Ensuring mathematical success for all*. National Council of Teachers of Mathematics, Incorporated.

[19] Yuhan Lin and David Weintrop. 2021. The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages* 67 (2021), 101075.

[20] Muhsin Menekse. 2015. Computer science teacher professional development in the United States: a review of studies published between 2004 and 2014. *Computer Science Education* 25, 4 (2015), 325–350.

[21] National Governors Assocation Center for Best Practices & Council of Chief State School Officers. 2010. Common Core State Standards for Mathematics. https://learning.ccsso.org/wp-content/uploads/2022/11/Math_Standards1.pdf

[22] Wing Shui Ng. 2017. Coding education for kids: What to learn? How to prepare teachers. *Proceedings of ICICTE* (2017), 195–205.

[23] Maria Nicolaidou and George Philippou. 2003. Attitudes towards mathematics, self-efficacy and achievement in problem solving. *European research in mathematics education III* 1, 11 (2003).

[24] National Council of Teachers of Mathematics. 2000. Principles and standards for school mathematics. *Reston, VA: National Council of Teachers of Mathematics* (2000).

[25] Hyejin Park, Tuğba Boz, Amanda Sawyer, and James C Willingham. 2023. Triangle explorations and constructions using robots. *Mathematics Teacher: Learning and Teaching PK-12* 116, 5 (2023), 392–398.

[26] Dominic Peressini, Hilda Borko, Lew Romagnano, Eric Knuth, and Christine Willis. 2004. A conceptual framework for learning to teach secondary mathematics: A situative perspective. *Educational Studies in Mathematics* 56 (2004), 67–96.

[27] Kathryn M Rich, Aman Yadav, and Christina V Schwarz. 2019. Computational thinking, mathematics, and science: Elementary teachers' perspectives on integration. *Journal of Technology and Teacher Education* 27, 2 (2019), 165–205.

[28] José Antonio Rodríguez-Martínez, José Antonio González-Calero, and José Manuel Sáez-López. 2020. Computational thinking and mathematics using Scratch: an experiment with sixth-grade students. *Interactive Learning Environments* 28, 3 (2020), 316–327.

[29] Megan Staples and Jill Newton. 2016. Teachers' contextualization of argumentation in the mathematics classroom. *Theory into Practice* 55, 4 (2016), 294–301.

[30] Henrik Stigberg and Susanne Stigberg. 2020. Teaching programming and mathematics in practice: A case study from a Swedish primary school. *Policy futures in education* 18, 4 (2020), 483–496.

[31] Peter Vinnervik. 2022. Implementing programming in school mathematics and technology: teachers' intrinsic and extrinsic challenges. *International journal of technology and design education* 32, 1 (2022), 213–242.

[32] George Watson. 2006. Technology professional development: Long-term effects on teacher self-efficacy. *Journal of Technology and Teacher Education* 14, 1 (2006), 151–166.

[33] Erna Yackel and Paul Cobb. 1996. Sociomathematical norms, argumentation, and autonomy in mathematics. *Journal for research in mathematics education* 27, 4 (1996), 458–477.