Infusing Computing: Analyzing Teacher Programming Products in K-12 Computational Thinking Professional Development

Yihuan Dong, Veronica Cateté, Nicholas Lytle, Amy Isvik, Tiffany Barnes
Robin Jocius, Jennifer Albert, Deepti Joshi, Richard Robinson, Ashley Andrews
North Carolina State University, Raleigh, North Carolina
The Citadel, Charleston, South Carolina
{ydong2,vmcatete,nalytle,aaisvik,tmbarnes}@ncsu.edu,{rjocius,jalbert,djoshi,rjmr,ashley.andrews}@citadel.edu

ABSTRACT

In summer 2018, we conducted two week-long professional development workshops for 116 middle and high school teachers interested in infusing computational thinking (CT) into their classrooms. Teachers learned to program in Snap*!*, connect CT to their disciplines, and create infused CT learning segments for their classes. This paper investigates the extent to which teachers were able to successfully infuse CT skills of pattern recognition, abstraction, decomposition, and algorithms into their learning products.

In this work, we analyzed 58 teacher-designed programming products to look for common characteristics, such as project type, intended coding requirements for their students, and code features/functionality. Teacher-created products were classified into five types: animation, interactive story, quiz, intended game, and simulation/exploration tools. Coding requirements varied from using and/or explaining provided code, modifying existing code, programming with starter code, to building entire programs. Products were classified according to the extent to which they involved sprite manipulation, questions/answers, event handling, drawing, and control blocks. We found that teachers from different disciplines created products that vary in type, coding requirements, and features to suit their specific needs. Moreover, we found relationships between discipline, project type, and the required coding teachers expected students to do.

Our results inform future Infusing Computing Professional Development (PD) to provide more targeted training to support different teacher needs.

CCS CONCEPTS

• **Social and professional topics** → **Computational thinking**; *K-12 education*.

KEYWORDS

Professional Development; Computational Thinking; Programming; K-12

ITiCSE '19, July 15–17, 2019, Aberdeen, Scotland Uk

ACM ISBN 978-1-4503-6301-3/19/07. . . $15.00
https://doi.org/10.1145/3304221.3319772

ACM Reference Format:
Yihuan Dong, Veronica Cateté, Nicholas Lytle, Amy Isvik, Tiffany Barnes and Robin Jocius, Jennifer Albert, Deepti Joshi, Richard Robinson, Ashley Andrews. 2019. Infusing Computing: Analyzing Teacher Programming Products in K-12 Computational Thinking Professional Development. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland Uk.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3304221.3319772

1 INTRODUCTION

As computers have become an integral part of our life and work, it is widely agreed that helping students master computer science (CS) and computational thinking (CT) skills at an early age may help them become more competitive in the future workforce [8]. One way to broaden participation in computer science is to infuse computational thinking into daily K-12 classroom teaching [17]. However, since the majority of K-12 teachers have no prior training in computer science or computational thinking practices and skills [10], they need to be properly trained in order to design and teach CT-rich curricula.

Researchers have designed a number of professional development (PD) workshops and initiatives to equip teachers with the skills and tools to teach their students computer science and computational thinking [11, 13]. Programming has played an important role in many of these professional development workshops, because even though computational thinking can be taught without programming [4], learning to program can be a powerful tool to help elaborate computational thinking concepts, such as abstraction and algorithms, in a situated context. Programming can also enable teachers to create or modify existing programming artifacts in order to design curricular materials that infuse computational thinking with content-specific practices and skills.

To successfully infuse programming into K-12 classrooms, it is important to help teachers align programming with their subject learning goals [10]. This requires developing knowledge of what types of projects the teachers could use to align CT-infused activities with existing standards and curricula and to identify the kinds of coding activities the teachers design to support student learning. However, little is known about how teachers intend to infuse programming activities into their existing curricula.

In this work, we analyzed the teacher programming products collected from two computational thinking themed professional workshops for middle and high school teachers, guided by the following research questions:

- What project types were found in the teacher products?
- How much coding was designed for the students to do?

- What were the common features the teachers used to create their programs?
- How did the teachers connect coding activities to the CT elements?

We wish to help researchers understand middle and high school teachers' needs regarding the use of programming to teach CT in their domain. In addition, our findings can inform the improvement of programming elements in professional development training to become more targeted based on the teachers' varied needs.

2 LITERATURE REVIEW

Research shows that teacher professional development is critical to any successful change in educational practice [14], including changes related to computational thinking integration into K-12 schools [2, 3, 18]. Specifically, Barr and Stevenson (2011) identified two major areas of need in relation to teacher PD in CT: (1) a clear definition of what CT is and how it applies to students and content, and (2) explicit, ongoing training and support for K-12 teachers.

In order to realize CT as an essential component of the curriculum across disciplinary areas, teachers must integrate "activities that make visible the inherent overlap of CT ideas and practices with subject area concepts" that allow students to engage in hands-on concepts and learning [17]. However, teachers must first understand what CT is, how it connects to their curricula, and how it might support students' understandings of content.

We used the simplified PRADA framework [9] to help teachers understand computing concepts and recognize the CT elements that already exist in their curriculum. In PRADA, the focus is on Pattern Recognition, Abstraction, Decomposition, and Algorithms, we use these keywords to connect key CT elements to teachers' disciplines (e.g. a function in math is very similar to an algorithm). Additionally, we engage teachers in hands-on coding sessions to learn basic CT and programming skills, helping teachers to create their own Snap*!* projects.

Based on this design, we seek to discover ways that teacher programs reflect their understanding of CT and how their students might learn CT when doing the created activities. There have been attempts to evaluate student block-based programming projects to measure CT learning and programming outcomes for students. These studies categorizes student programming projects into types (e.g. games, music videos, storytelling, etc) and compare how these types of projects motivate students to use desired programming concepts, such as loops, conditionals, and variables [1, 5, 7]. Other research connects programming concepts with CT elements to measure how much CT is present in student projects. For example, Dr. Scratch[12] performs static analysis on Scratch projects to identify the presence or absence of key CT competencies such as parallelism, abstraction, and flow control.

Analyzing student programming artifacts helps identify what CT and programming concepts students can demonstrate. While the teachers participating in CT themed PD are similar to the students in terms of programming skill and prior knowledge, teacher-created projects may have a different focus than student projects, because teachers have purposefully designed their projects to serve as a tool for teaching both the subject knowledge and the CT concepts. Thus, analyzing teacher projects can provide insight into what CT and programming concepts the teachers value and how they relate them to their disciplines.

3 METHOD

We collected data from two intensive, five-day Infusing Computing PD workshops [1] in North and South Carolina in Summer 2018, during which we engaged 116 middle and high school teachers in designing plans to infuse computational thinking into their classrooms. According to our survey result prior to the PD, 48% of the participating teachers never had any programming experience, 22% had their students do hour of code activities, 28% had some but not extensive prior programming experience(e.g. HTML, JavaScript, Scratch, or code.org), and 2% claimed to be proficient in at least one programming language. We designed a 3C (Code, Connect, Create) model to structure the PD and designed each day to include sessions for each of these three elements. The Code sessions introduced basic concepts and operations in the Snap*!* programming environment [6] (e.g. sprites, blocks, drag and drop operations), control structures (e.g. loops, conditionals, variables), and lists using simple programming examples and finally culminating in practice with a simulation developed to integrate CT into science classes. In the discipline-specific Connect sessions, we introduced the PRADA elements and helped teachers identify the CT concepts that already exist or can be easily infused into their own content areas. The Create session allowed the teachers to work individually or to collaborate in teams to develop their CT-infused learning segments along with a Snap*!* program for use in their classrooms.

Table 1: The number of teacher products and project files by content area

	Science	Math	Inter-disciplinary	Humani-ties	Total
Products	18	11	4	7	40
Proj. Files	25	13	7	13	58

The teacher product analyzed in this work includes a learning segment document and a Snap*!* project file that teachers submitted at the end of the PD. Analysis of the learning segments allows us to understand teachers' intentions for their CT-infused lessons. Some of the items in the learning segments include: the disciplinary learning goal, the CT learning goal, the PRADA elements involved, the activities planned (with and/or without programming) for the students, and the end-product and evaluation criteria for grading. We required the teachers to do some programming and submit a Snap*!* project file so that they had experience with the types of programming concepts and practices their students could use in their classrooms. The Snap*!* project can either be a model for the type of project students would be expected to create themselves or a project that teachers created for the students to play with and explore. Teachers who worked in groups were asked to submit at least one Snap*!* project file per group, but they could submit more than one program if appropriate for their learning segment. Before analysis, we removed one product from our analysis for failing to submit their learning segment document, leaving us no way to

[1] For more information about the PD, visit https://www.infusingcomputing.com/

know how the teachers planned to use their Snap*!* project file in the class. In addition, we removed 4 project files for being starter code (that was repetitive with the corresponding complete project) and removed one duplicate project file. In total, we analyzed 40 teacher product submissions comprised of 40 learning segments and 58 Snap*!* project files, as is shown in Table 1.

We analyzed the teacher products at two levels: a high-level content analysis and a low-level automated programming concept analysis. Four of the authors performed a two-phase content analysis to identify the type of products, the programming features in the project files, and the coding activities designed for students. These raters were all CS education graduate students with experience designing, implementing, and evaluating CT curriculum for middle school classrooms. In the first phase, each rater looked through all the learning segments and the Snap*!* project files and took notes on the type, features, and the student coding requirements for each teacher project. The raters were given instructions on what to look for in general but were also encouraged to include anything that seemed relevant and/or uncertain for later discussion. In the second phase, the first author formed an initial categorization of the teacher products based on all the rater notes. Then, all four raters discussed each category definition until they reached full agreement. Since the type categories were more ambiguous, the four raters collaboratively determined the final type category for each project. For the other more straightforward characteristics, all the teacher projects were re-coded by the first author based on the final definition. Due to the design of our data collection instrument – it only provided small text boxes to indicate which PRADA elements related to their products – many teachers did not explain the relationship between their coding activity and their plans for teaching computational thinking. Thus, to identify how the teachers related the coding activities to their plans for teaching computational thinking, the raters manually read through the learning segments and extracted excerpts that suggest teachers' understanding of the connection between coding and CT.

The second analysis is a programming concept analysis. We created an *analysis program* that takes in the Snap*!* project files and counts if and how many times some blocks appeared, which may indicate teachers' level of understanding and ability to use certain programming concepts. The output of the analysis program includes the number of times each default or imported block is used, the number of variables (both local and global) created and used, and the number of custom (teacher-created) blocks created and used. This analysis can give us a general glimpse of how well the teachers were able to apply what they learned from the coding session to their project. The statistics also shed light on some findings in the content analysis as discussed below.

4 RESULTS

4.1 Content Analysis

4.1.1 Project Type. The raters identified five different project types in the teacher created project files:

(1) **Animation**: A non-interactive instructive, introductive, or model video
(2) **Interactive Storyboard**: A linear or non-linear animation with non-consequential interactions and a connected series of scenes
(3) **Quiz**: A project that asks students a series of questions with the sole intent to test students' knowledge about a subject
(4) **Intended Game**: A project intended to be made into a game by the teachers, even though the project may be non-game-like on submission
(5) **Exploration/Simulation Tools**: A project that provides a tool for the students to interactively explore and experiment with a concept

Figure 1 shows the number of teacher products of each type. Looking at the total number, we can see that more than a third of the products are Simulation/Exploration Tools. Animation and Interactive Storyboards together make up half of the teacher products. There are fewer Quizzes and Intended Games than the other types. Looking at the content area of each type, we find that the humanities and interdisciplinary projects are mostly Animation and Interactive Stories, whereas the Math products are mostly Intended Game and Simulation Tools. Science has about half of its projects in Simulation/Exploration Tools category while the rest of its projects were distributed roughly evenly among Animation, Interactive Storyboard, and Quiz, suggesting that science teachers may have more diverse uses for programs in their lessons.

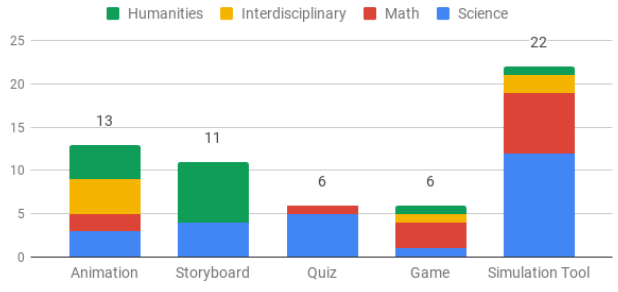

Figure 1: Number of projects in each project type

4.1.2 Student Coding Requirements. Many learning segments described the coding students would do to complete the activity. Some learning segments did not describe the coding requirement explicitly, but it could be inferred from the description of the activities. For example, a team of middle school science teachers first said "the students will be required to write their own code..." and then said "once they have created an account [in Netsblox], we will share the program with them... for themselves and their partners to manipulate", indicating the students need to write a program using a starter code. Six learning segments did not mention student coding activities at all, thus were given the 'unspecified' tag.

Raters identified four categories of student coding requirements from the teacher products. These coding requirements varied from using and/or explaining provided code, modifying existing code, programming with starter code, to building entire programs. For using and/or explaining provided code, the teachers planned to give the students a completed program for them to play and explore, sometimes accompanied by activities that require explaining the

code in the program, without hands-on coding. Modifying existing code requires the students to play and make changes to the program to adapt to new information. For programming with starter code, the teachers would provide students with a starter project file that already has some code created and require students to either finish it or extend its functionalities. Building entire programs would require students to learn the requirements of the full project, usually through playing with a completed program, and then creating their own project from scratch.

The coding requirements discovered seem consistent with the popular Use-Modify-Create practice for teaching youth CT concepts [10], indicating that the teachers are familiar with the practice of having students use a working program to become familiar with an idea, then modify one, and then create their own.

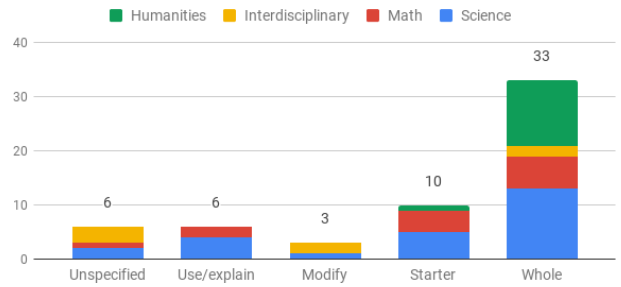

Figure 2: Number of projects with each coding requirement

Figure 2 shows the number of projects with each coding requirement. The result shows that more than half of the teacher projects tasked students with making the whole project from scratch.

4.1.3 Feature Analysis. The raters identified five common features (Sprite manipulation, QA, Event handling, Control, and Drawing) in the teacher projects. Figure 3 shows the percentage of the projects having each feature by project type. As expected, all 58 projects had some sort of Sprite Manipulation feature, which is the core mechanic and a major strength of the Snap*!* environment. Twenty-seven projects used Questions/Answers (QA), asking a question and checking answers. While QA was not present in the (non-interactive) animations, it was identified in all 6 of the quiz projects, and in over 50% of the interactive stories, games, and simulations, indicating that QA serves as an important type of interaction in the teacher-created Snap*!* projects. Two-thirds (40/58) of the projects involve Event Handling to synchronize behaviors between sprites. The most popular synchronization method (28/40) was by broadcasting a message to signal the next event. Three-quarters of projects (44/58) included basic programming control structures, such as if-then conditions, game loops, and randomized events. Animation having the lowest proportion using control structures, which makes sense given that most of the animations have a single, chronological plot that does not repeat or deviate based off of input conditions. Those animations with control structures usually used loops to draw shapes, corresponding to the DrawSquare and RowOfHouses programs introduced in the PD's Code session. Finally, 20 of the 58 projects used Drawing features.

Interestingly, we found that some teachers used the doWait block instead of message broadcasting for synchronization purposes. These teachers asked a sprite to wait a pre-set number of seconds for other sprites to complete their actions before continuing to the next action. This discovery is consistent with another research that studied creative products from elementary school students [16]. While using doWait block is one way to achieve synchronization, it is not a preferred method in many cases because of its static nature – one has to manually adjust the wait time for every sprite in the program to adapt to new information. This finding is one of the many examples that suggests problems that the Code session could easily address to help the teachers make certain features more efficiently.

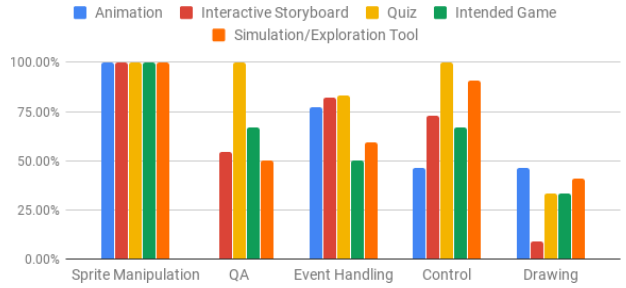

Figure 3: The percentage of the projects that have each feature by project type

4.1.4 CT in Coding Activities. We analyzed how teachers connected the coding activities to each element in PRADA by discussing what connection the teachers made with an excerpt from the learning segment that demonstrates the connection.

Pattern recognition does not seem to be a difficult element for the teachers to understand and relate to, as the teachers who identified pattern recognition in the coding activity generally made the correct connection. Teachers were able to recognize patterns in the code from two aspects. One aspect is having repeated code in a piece of script, linking it to a repeat block ("Using a repeat in the code to be able to use the 4 nitrogenous bases of DNA limitlessly"). The other aspect is having duplicate code between sprites ("6 regions that have the same code, and repetition within the code itself").

Decomposition in coding activities happens mostly when explaining and understanding existing code, often accompanied by making abstractions of decomposed code chunks ("Break down plot diagram into subcategories and look further into what blocks go into a specific abstraction"). Some teachers discussed decomposition in a more subtle way by considering the conditionals and loops as means to decompose an algorithm ("if' and 'repeat' blocks are used in the code to show a breakdown of the parts of the algorithm").

Many science and math teachers were able to identify different kinds of abstraction in the code. Many teachers were able to link the definition of custom blocks to abstraction (e.g. "Name these repeated steps [of calculating the roots and the vertex of a binary equation] by calling the process Find Roots and Find Vertex"). Some teachers recognized that the use of variables is also a kind of abstraction ("Variables have been created for specific [geological] values"). In

contrast, abstraction seemed to be a more challenging concept for the interdisciplinary groups and humanities teachers to apply to their projects, as none of them mentioned abstraction in their coding activities. This finding suggests that more support is needed for the non-STEM teachers to conceptualize and apply abstraction in programming to their subject-specific context.

Similar to abstraction, algorithms were discussed only by science and math teachers. Some science and math teachers were able to relate algorithms to the "the code used in the script". These learning segments would require the students to "use an algorithm to create a block of code that will ask a question and respond to the answer as to whether correct or incorrect". A group of math teachers took a step further and noted the generalizability of an algorithm, stating that "Transcription is a step by step process and the code that is created demonstrating transcription can be used with any sequence of DNA nitrogenous bases to create an mRNA transcript." None of the interdisciplinary/humanities teachers mentioned how algorithms connect to coding in their learning segments.

4.2 Programming Concept Analysis

Table 2 shows the results of the programming concept analysis. In general, around two-thirds of the projects used loops and conditionals (If), but the percentage of projects having loops and conditionals varies greatly by type. This corroborates the finding in the content analysis where few Animation projects have Control features. Both Custom blocks and Variables appeared in about half of the projects, suggesting that these concepts are still hard for many teachers to apply. In addition, even though being a very useful data structure, lists may be difficult for the teachers to learn as it only appeared in 1/3 of the projects. Even fewer projects (13.79%) used advanced list operations like insertion or deletion.

Table 2: The number of projects that have each programming concepts by project type

	Loop	If	Custom	Var	List
Anim (13)	7(51%)	2(15%)	5(38%)	5(38%)	6(46%)
Story (11)	7(64%)	10(91%)	7(64%)	4(36%)	0(0%)
Quiz (6)	6(100%)	5(83%)	2(33%)	3(50%)	2(33%)
Game (6)	3(50%)	4(67%)	3(50%)	4(67%)	0(0%)
Sim (22)	15(68%)	20(91%)	14(64%)	18(82%)	11(50%)
Total	38(66%)	41(71%)	31(53%)	34(59%)	19(33%)

The programming concept analysis helps expose some potential issues in the teachers' understanding of programming concepts. For example, as experienced programmers we can see that the QA feature could be easily abstracted into a custom block that uses lists to manage questions and their corresponding solutions. However, only two out of six Quiz projects used custom blocks and lists. This could suggest that the teachers may have failed to recognize this abstraction or that they're simply not comfortable with using custom blocks and/or lists. Moreover, programming concept analysis helped identify the nature of the types. Animation and Interactive Storyboard projects had lower percentages of variable usage, which is most likely due to the fact that neither project type typically requires storing information for later use.

The Intended Game type scored a relatively low percentage on all programming concepts. This is also consistent with the result in the Feature analysis shown in Figure 3. Upon close inspection, we find that this could be due to the different mechanics of the games. The games that rely on mouse click-and-drag events tend to use fewer desired CT/programming concepts than others since their logic is often simply responding to cursor events. This suggests that teachers who want to make games should think about how their game mechanics relate to the programming/CT concepts they want students to learn.

5 DISCUSSION

To further investigate the coding activities in the teacher projects, we plotted the project type vs. coding requirements as shown in Figure 4. We arranged the project types according to CT/programming difficulty and the coding requirements according to how much coding students would do. Project types are arranged from left to right from least difficult to most difficult to infuse CT/programming concepts into if implemented correctly. Coding requirements range from the least (no coding) at the bottom to the most (whole project) at the top. Figure 4 shows the distribution of project types versus coding requirements for each content area with the size of the dot mapping to the number of projects at that spot.

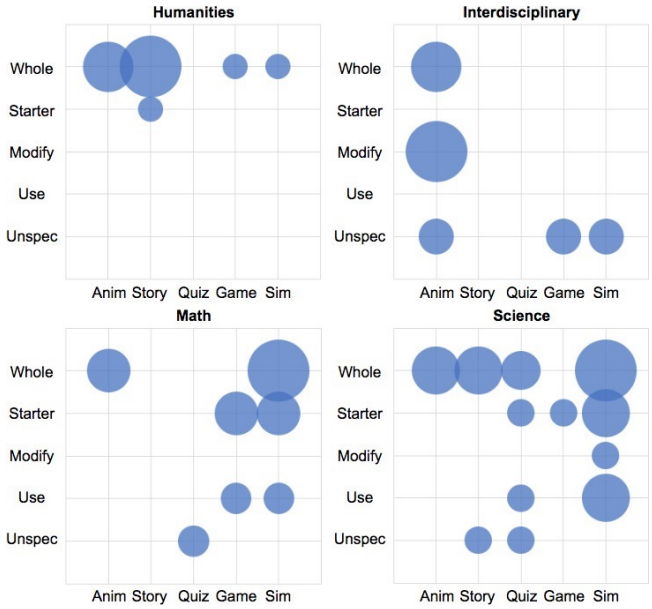

Figure 4: The relationship between project type and coding requirement by content area

These plots reveal tendencies for the teacher-created projects affiliated with each discipline. Specifically, the humanities projects lean more towards building animations and require students to do more coding. We think this is possibly due to the fact that humanities classrooms often require students to write or create content. In addition, as suggested by our product analysis, animations and interactive storyboards tend to have less complex logic and are relatively easy to make, so these teachers may feel more confident in their students' ability to make the whole project.

Math projects tend to be simulation/exploration based and expect students to do more coding. This could be because that many mathematical concepts, norms, practices, and vocabulary terms are in parallel with programming concepts [15]. For instance, the steps taken to solve a math problem can be easily transformed into an algorithm. By turning math problems like finding the best rates and exploring transformations into making simulation/exploration tools, students gain the opportunity to develop a deeper understanding of the concepts through hands-on programming.

Science projects are more varied in terms of project type and coding requirement, which may indicate that the science teachers have a larger variety of needs for their projects. For example, a group of science teachers created a curriculum to teach the Energy Pyramid. Their product included four project files: one project file is an interactive storyboard that introduces the food chain, one project file is a simulation that explores the population dynamics between the predator and the prey, and the other two project files are quizzes, one testing students' knowledge of the energy transformation in the energy pyramid, and one testing on the food chain. These varied needs are also reflected in the coding activities designed for students. For simpler concepts or phenomena, the teacher may want the students to create the whole program to increase understanding of the underlying mechanic. However, for more complex concepts, the teacher may simply want the students to use the project as a tool for the students to explore the pattern behind the phenomena.

By showing the disciplinary differences in terms of project type and coding requirements, we're not trying to argue which type or how much coding is good or bad. Rather, using the findings from the teacher product analysis, we hope to better understand the needs of the teachers from different disciplines in order to help them find better, more tailored ways to infuse programming and computational thinking into their curricula. For example, our findings show that humanities teachers are fond of using animation and interactive storyboards in their lessons but are having trouble identifying and infusing some CT elements and programming concepts into their products. A potential solution would be introducing simple modeling concepts to the teachers that transform their animations from heavily relying on the wait blocks to using variables and event handling to govern the chronological progression of the story. Other solutions may include showing the teachers diverse examples and types of coding activities that could potentially be used to teach the same knowledge.

In addition, the feature analysis and programming concept analysis not only tell us how much programming the teachers have learned, but also suggest places where we can provide more scaffolding to help the teachers be more successful in project development. For example, by better understanding what features teachers often need in their projects, we can provide more targeted scaffolding by creating pre-made features and example projects that the teachers can easily adapt and use in their projects, such as a QA block. It's important to understand and remember that these teachers are not learning programming to become professional programmers. Rather, they only need to concentrate on the elements of programming that afford CT and connect to their domain, standards, and curriculum. Thus, by giving the teachers pre-made features and examples during the Code sessions, we are not only giving them the chance to practice explaining code and examining how the features are made in the desired ways, but also relieving the teachers from the struggle of creating certain features so they can spend time on designing learning experiences for their lessons.

One limitation of this work is that the project files submitted weren't necessarily refined products ready to be used, but were mostly prototypes to showcase what the teachers intended for their students to do in their classrooms. Thus, certain features that could demonstrate teachers' understanding of programming concepts and CT elements may not have been completed. However, while the prototype characteristics were a limitation, they also afford insight into what the teachers were able to learn and make in a week of training, which might shed light on how much programming they feel comfortable teaching after a short-term training and how much they believe their students are able to learn in these lessons. Another limitation might be the fact that the coding sessions were taught mostly with create activities, which might affect the choice of the coding requirement teachers designed in their learning segments.

6 CONCLUSION

In this paper, we analyzed 58 middle and high school teacher-designed programming projects from two computational thinking PD workshops to help us understand how the K-12 teachers intend to use programming to teach CT in their classrooms.

From our analysis, we identified five project types in the teacher programming projects: animation, interactive storyboard, quiz, intended game, and simulation/exploration tool. The teachers generally have four kinds of coding requirements in their lesson plans: using and/or explaining provided code, modifying existing code, programming with starter code, and building entire programs. The common features in the teacher programming projects can be classified into five categories: sprite manipulation, QA, event handling, control, and drawing. Humanities and interdisciplinary products were mostly animations and interactive storyboards and required the students to code the whole project. The majority of the math projects were simulation/exploration tools and intended games that required students to either code the whole project from scratch or with starter code. The classification of the science products varied in project type and coding requirements, depending on the purpose of the project. Regarding connecting coding activities to computational thinking, many science and math teachers were able to make reasonable connections. However, humanities and interdisciplinary teachers appeared to have trouble identifying abstraction and algorithms in their planned coding activities.

Our work provides insights on how the teachers intend to use programming to teach computational thinking in their own content area lessons. These insights can help us provide more scaffolding for the teachers by making relevant example programs and pre-made custom blocks to help the teachers develop their projects more efficiently. We will use these pre-made examples in the Code sessions, modeling content that suits teachers' needs while also better conveying CT concepts that can be integrated and learned by students in their classrooms.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grant numbers 1742351 and 1742332.

REFERENCES

[1] Joel C Adams and Andrew R Webster. 2012. What do students learn about programming from game, music video, and storytelling projects?. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 643–648.

[2] Charoula Angeli, Joke Voogt, Andrew Fluck, Mary Webb, Margaret Cox, Joyce Malyn-Smith, and Jason Zagami. 2016. A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society* 19, 3 (2016).

[3] Valerie Barr and Chris Stephenson. 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 (2011), 48–54.

[4] Tim Bell, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology* 13, 1 (2009), 20–29.

[5] Alexandra Funke, Katharina Geldreich, and Peter Hubwieser. 2017. Analysis of scratch projects of an introductory programming course for primary school students. In *Global Engineering Education Conference (EDUCON), 2017 IEEE*. IEEE, 1229–1236.

[6] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The beauty and joy of computing. *ACM Inroads* 6, 4 (2015), 71–79.

[7] Shuchi Grover, Satabdi Basu, and Patricia Schank. 2018. What We Can Learn About Student Learning From Open-Ended Programming Projects in Middle School Computer Science. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 999–1004.

[8] Shuchi Grover and Roy Pea. 2013. Computational thinking in K–12: A review of the state of the field. *Educational Researcher* 42, 1 (2013), 38–43.

[9] Author Hidden. XXXX. Hidden Paper. In *Anonymous*.

[10] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *Acm Inroads* 2, 1 (2011), 32–37.

[11] María Cecilia Martinez, Marcos J. Gomez, Marco Moresi, and Luciana Benotti. 2016. Lessons Learned on Computer Science Teachers Professional Development. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 77–82. https://doi.org/10.1145/2899415.2899460

[12] Jesús Moreno-León and Gregorio Robles. 2015. Dr. Scratch: A web tool to automatically evaluate Scratch projects. In *Proceedings of the workshop in primary and secondary computing education*. ACM, 132–133.

[13] Jennifer Rosato, Chery Lucarelli, Cassandra Beckworth, and Ralph Morelli. 2017. A Comparison of Online and Hybrid Professional Development for CS Principles Teachers. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. ACM, New York, NY, USA, 140–145. https://doi.org/10.1145/3059009.3059060

[14] Lee S Shulman and Judith H Shulman. 2004. How and what teachers learn: A shifting perspective. *Journal of curriculum studies* 36, 2 (2004), 257–271.

[15] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. 2016. Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology* 25, 1 (2016), 127–147.

[16] David Weintrop, Alexandria K Hansen, Danielle B Harlow, and Diana Franklin. 2018. Starting from Scratch: Outcomes of early computer science learning experiences and implications for what comes next. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM, 142–150.

[17] Aman Yadav, Hai Hong, and Chris Stephenson. 2016. Computational thinking for all: pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends* 60, 6 (2016), 565–568.

[18] Aman Yadav, Chris Mayfield, Ninger Zhou, Susanne Hambrusch, and John T Korb. 2014. Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education (TOCE)* 14, 1 (2014), 5.