Boston College
Lynch School of Education

Department of
Teacher Education, Special Education, and Curriculum & Instruction

Curriculum and Instruction

EXTENDING TEXT-BASED PROGRAMMING LANGUAGES TO EMBED COMPUTING INTO MIDDLE SCHOOL SCIENCE CLASSROOMS

Dissertation
by

YANG XU

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

December 2019

EXTENDING TEXT-BASED PROGRAMMING LANGUAGES TO EMBED COMPUTING INTO MIDDLE SCHOOL SCIENCE CLASSROOMS

Yang Xu

Dissertation Chair: Dr. C. Patrick Proctor

Abstract

The demand for talent in the technology sector and the notion of computational thinking as everyday skills propel computing to enter middle school classrooms. The growing popularity of physical computing in educational spaces also infuses computing with elements of creativity and joy. Despite these recent movements, computing remains primarily in informal spaces due to a shortage of computer science teachers and the increasing focus on standardized testing. Arguing that computing and science share practices, this study views computing as problem-solving tools for science and proposes an integrated approach to teaching computing in science classrooms that takes advantage of the affordances of modern physical computing devices. Based on this perspective, a set of physical computing tools was developed to de-emphasize the mechanisms of computer science and shift focus to problem-solving and authentic scientific practices. This study aims to investigate the experiences of two science teachers and 16 students who learned to build self-regulated smart tabletop greenhouses with these tools as complete novices and critically evaluate the principles that undergird the design of the tools.

With a qualitative, multiple case study design, this study answers two questions: 1) how did the teachers implement and reflect on their instruction? 2) how did the students engage with computing and science? Data from interviews and observations suggest that although the teachers shared similar instructional practices, their

conceptualizations of the interplay between computing and science differed initially. They also had different instructional focuses and followed different trajectories in teaching, which may have produced subtly different understandings of computing-science relationships from their students. Despite these differences, all participants' understandings of computing-science relationship conformed to a reciprocal pattern, which augmented the shared-practice argument for the integrated approach found in the literature. The challenges that the participants experienced contributed to the revision of the design of the computing tools. Based on these findings, the study recommends future directions in disambiguating the role of computing in middle school classrooms and in working with science teachers who are often simultaneously content experts and computing novices.

This dissertation is dedicated to the late Tony L. Gasparich, without whom none of my journey in the United States would have been possible, and to all of the teachers and students who helped me develop the prototype of the computing tools in this study.

Acknowledgements

I would like to give my most heartfelt thanks to my dissertation committee, Drs. Patrick Proctor (Chair), Michael Barnett, and Katherine McNeill, for your unparalleled guidance and support during not only my dissertation process but also my entire time as a PhD student at Boston College. Each of you had great influence on the trajectory of my studies. You truly are the best committee ever!

To Dr. Proctor, who is also my advisor, thank you for supporting me whole-heartedly in my pursuit of a career in data science and for directing me back on track when I was lost and confused. I am truly blessed to have you as my advisor who is so genuinely invested in his students. Thank you Patrick!

To Dr. Barnett, thank you for the wonderful opportunity to learn and grow as a techie on your team and to be inspired by your ingenious ideas. You taught me how to think out of the box and look at the big picture, which will be helpful for the rest of my life. Thank you, Mike!

To Dr. McNeill, thank you for giving me the best advice and helping me make the best decision of my life! Your feedback on my work is always spot-on, and your learning sciences class is still the best course I have ever taken. Thank you, Kate!

I would also like to thank my cohort: Allison Nannemann, Anne Vera Cruz, Anna Noble Dunphy, Caitlin Malloy, Chris Chang-Bacon, Juan Cristóbal García-Huidobro S.J., Kate Soules, Paul Madden, Qianqian Zhang-Wu, Renata Love Jones, Shaneé Washington-Wangia. You each helped me at different points of my journey. Although we are now scattered all over the country, we will always be #BestCohortEver!

Thank you, Dr. Samantha Daley, whose mentorship and support helped me learn and grow at CAST, and shout out to Becca Louick, my CAST buddy, for your help and support in my first years!

Thank you to my wonderful friends and colleagues: Raj Rupani, Jasmine Alvarado, Dave Jackson, Megan McKinley-Hicks, Ariella Suchow, Helen Zhang, Yihong Cheng, Haerin Park, Jordan Lawson, Kevin Mader, Caroline Vuilleumier, Kirsten Rene, and Caitlin Long! You guys are the best!

Special thanks to my new colleagues at The Policy Lab at Brown University, David Yokum, Kevin Wilson, Attiyya Houston, and Matthew Lyddon, for helping me with my defense!

Last but not least, thank you, Paul Tate, for your unconditional support during this entire process. I would not have been able to do this without your help and encouragement!

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Background and Research Problem

Computing is constantly undergoing rapid changes. The formidable computing power of modern computers has fueled the explosive development of artificial intelligence (AI) which has the potential of reshaping the world we live in and completely altering the landscape of the labor market (Grover & Pea, 2017). A White House report (Office of Science and Technology Policy (OSTP), 2016) predicted that AI-powered robots and computing devices will continue to displace more low-skill jobs that can easily be automated to create more opportunities favoring skilled workers with command of the very technologies that undergird AI, such as "big data," machine learning, and cloud computing. Warning that this downward pressure could deepen income inequality in the United States, the report highlighted the importance of creating a more substantial and highly diverse AI workforce and preparing American youth for the future job market by providing them with a high-quality education in STEM fields, especially in computer science.

In addition to the call from the labor force, computer science educators are now arguing that engaging in computing could cultivate "computational thinking" skills that foster a paradigm of thinking rather than merely developing programming skills. Jeannette Wing (2006, 2008) coined this overarching term to refer to a set of loosely-coupled skills that involve "solving problems, designing systems, and understanding human behaviors, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33). This concept is also considered as a continuation and revitalization of the pioneering work of Seymour Papert (1980, 1993) who vocally advocated the use of

computer science to teach children to think. Since its inception, computational thinking has undergone much scrutiny and clarification (e.g. Wing, 2008; Aho, 2008; Denning, 2009), mainly revolving what constitutes "computational thinking" (Grover & Pea, 2013) and how it differs from other critical skills such as "mathematical thinking" and "algorithmic thinking." However, the idea that computational thinking is "a fundamental skill for everyone, not just for computer scientists" (Wing, 2006, p. 33) has become widely accepted, and the STEM standards of many states, including the Massachusetts State STEM Standards, are recommending the inclusion of computational thinking in K-12 curriculum.

Contrasting the rising demand for skilled workers from the industry and growing enthusiasm about computational thinking in the academic sphere is the limited access to computing-related courses at the K-12 level. It is estimated that only around 10% - 25% of US high schools are offering computer science courses (Marder & Hughes, 2017; Guzdial, 2012). The challenge is multifold. According to a multiyear report by Gallup and Google (2016), while the majority of surveyed students, teachers, principals, and superintendents perceive computer science as important, administrators did not consider computer science a priority, as well as the shortage of qualified computer science teachers and the absence of testing requirements. Teachers, despite their interest in computer science, felt a lack of support from their superiors. Students, as a result, accessed learning resources to computer science through school-sponsored clubs and computer science incorporated in other classes. Unfortunately, unless systematic reform radically changes the test-driven climate of US schools, computing will remain a low priority for school administrators.

In addition to continuing the efforts of pushing computer science into K-12 schools, one possible avenue to broadening access to computer science could be to incorporate it into other courses. As scholarly debates continue to clarify the differences between computer science, practices of computing, and computational thinking, it becomes clear that it is computational thinking rather than computer science that is the desirable skill for every student to develop at the pre-college level (Aho, 2012, Wing, 2008, Grover & Pea, 2013). Moreover, computational thinking does not necessarily have to be developed in a computer science course especially since research has pointed out that teaching programming does not automatically lead to computational thinking (Grover & Pea, 2017; Wing, 2006). Arguably, with an optimal combination of tools and instruction that introduces practices of computing into the classroom, computational thinking can be synergistically integrated into other classes to augment the learning of other STEM disciplines.

Computing and Science

Given the argument above, this research study focuses on exploring the idea of introducing computing into K-12 science classrooms to develop computational thinking skills. Computers have been an indispensable tool for scientists ever since their inception. Astronomers have long been using computer simulations to precisely predict the positions of celestial bodies. The age of "big data" has fueled the exponential growth of computational science in that scientists now take advantage of the enormous computational power of modern computers and the gigantic volumes of data to pursue new frontiers in scientific inquires (Wing, 2006). The ability to use a computer to analyze data for scientific inquiry is a highly desirable, if not necessary, skill for today's scientists

(Machluf, Gelbart, Ben-dor & Yarden, 2015), a position which lends strong support for the integration of computers into science classrooms.

Although the use of computing has always been integral to authentic scientific inquiry, computing and science remain separate focuses in K-12 classrooms. Understandably, the computing devices scientists use are so specific to their fields that they are not appropriate for use in science classrooms (Hanauer et al., 2006). However, thanks to the five advancements in computing, it is now possible to approximate the authentic scientific processes in K-12 classrooms using modern computing devices:

1. Computing is becoming increasingly *mobile*. According to an IDC report (n.d.), laptops and tablets have continually encroached on the market share of desktop computers, and by 2022, over three times more laptops and tablets than desktops will be purchased. The market of smartphones, wearables, and connected smart home devices will continue to expand. As a result, computing no longer needs to be taught or performed in computer labs.

2. Computing is becoming increasingly *affordable*. Laptops such as Chromebooks and tablets such as Amazon Kindles are available for less than $100, while many inexpensive smartphones can be found everywhere. Raspberry Pi, a credit-card-sized computer designed for children to learn to code, is sold for about $35, while microcontroller boards, small computers designed for specific purposes, are even more affordable, thus further reducing entry threshold to computing.

3. Computing is becoming increasingly *physical*. Raspberry Pis and Arduinos provide accessible interfaces for students to control physical computing devices such as sensors and motors directly. Within a few lines of code, students can drive control LED

lights, turn on/off motors, and generate music with small speakers. The direct feedback from these physical devices provides educators space for creative instruction that scaffolds abstract concepts such as variables and loops, removing cognitive barriers and deepening understanding of otherwise tedious concepts.

4. Computing is becoming increasingly *connected*. Computing devices can connect with each other and to the Internet wirelessly which means that machines can exchange data more efficiently than ever. This trend gives rise to a host of innovative applications that were previously unimaginable, such as fitness trackers, smart home assistants, traffic monitoring, and real-time baggage tracking. As trillions of connected devices permeate the world we live in, the demand arises for a new generation of innovative computer scientists and engineers who are ready to employ the power of connected devices to reshape the landscape of computing and the broader society. For everyone else, a fundamental understanding of how to interact with connected devices could become a necessity and improve the quality of life (Kortuem, Bandara, Smith, Richards, & Petre, 2013).

5. Computing is becoming increasingly *approachable*. Block-based programming languages (BPLs) such as Scratch (Resnick et al, 2009) and Alice (Cooper, Dann, & Pausch, 2000) are bringing coding to young children, while text-based programming languages (TPLs) themselves are designed to be less arcane and more similar to natural languages, further reducing the difficulty in learning text-based programming languages.

Physical Computing in Science Classrooms

Physical computing is a relatively new approach to teaching computer science that embodies all these new trends. The core idea of this approach is to teach the students

programming by engaging them in the creative process of designing, building, and programming tangible and interactive objects that move, blink, and/or make noises with small, affordable computing devices such as micro-computers (Raspberry Pis), microcontrollers (Arduino, micro:bits), sensors, motors, and LED matrices (Przybylla & Romeike, 2014). The result of this powerful idea is engaging and enjoyable learning environments for concepts in programming that not only blend the virtual and physical world but also inspire creativity and innovation. Physical computing devices are also powerful instruments for authentic scientific inquiry in K-12 classrooms. They provide an extremely low-cost entrance for every student to experience computing, and their small sizes and portability are ideal for integration into daily science activities. Block-based programming languages and beginner-friendly text-based programming languages such as Python have removed barriers for students to program these devices for automated data collection, functionally mimicking how real-world scientists gather empirical data to answer their research questions. Connectivity with other devices enables student-collected data to be stored, transferred, processed, visualized, and analyzed for a better understanding of authentic scientific processes.

In the meantime, microcontrollers and sensors are strong candidates for developing computational thinking skills because of their focus on data (Dasgupta & Resnick, 2016). Students can even use data for automation, a core idea of computational thinking. For example, with a simple logical statement, the microcontroller could be instructed to turn on and off for temperature control, if the temperature sensor connected to it detects higher temperatures than a certain threshold. In this process of working with physical computing devices, more key ideas of computational thinking such as

algorithmic thinking and using abstraction (Grover & Pea, 2017; ISTE & CSTA, 2011; Lee et al., 2011) are embedded. In other words, students can develop computational thinking while using physical computing devices for scientific inquiry.

Despite the many educational opportunities that physical computing devices afford, introducing these devices into science classrooms without a prudent plan of execution can be as problematic as many failed technological integration projects in the past. Numerous challenges are currently present. First, educational physical computing is a blossoming world today with numerous software and hardware platforms, each designed for specific audiences while new platforms and technologies keep emerging that leave educators and researchers inundated with options. Second, it is difficult to strike a balance between science and computing. Given the socio-political atmosphere driven by standardized testing and accountability, the complexity of computing, which does not appear on standardized tests, might not impose additional burden for practitioners. Ideally, practices of computing should serve the dual purpose of improving science learning and fostering computational thinking, but these practices require considerable efforts and might draw the focus away from learning science. Most important is the lack of science teachers skilled at embedding practices of computing in science instruction, which currently is not part of science teacher training programs. These challenges highlight the need for a comprehensive solution that includes a curriculum and its accompanying tools and teacher professional development programs and accompanying design decisions carefully made to address each of these problems.

Research Purpose and Questions

This dissertation is a qualitative, multiple-case study that investigates the design

of one set of physical computing tools and the accompanying curriculum which aims to embed computing into scientific inquiry activities in middle school classrooms – the integrated approach to computing and science. The design supported the construction of an innovative learning environment in a formal setting where middle school students and their teachers learned coding and science together through physical computing, applying what they have learned to approach high-level, authentic inquiries in science and developing computational thinking skills. This study documents and reports the experiences of 16 eighth graders and their two science teachers in a Northeast US public school as they learn to design and build automated, smart, table-top greenhouses (Figure 1) with the said tools and curriculum. The purpose is to critically examine the principles that support the design of the computing tools and the curriculum and propose guidelines and optimal practices for success in embedding computing into science classrooms. The research questions are:

Figure 1. The GrowThings smart greenhouse.

1. How did the teachers implement and reflect on their instruction in this learning environment?
    a. How did they understand the interplay between computing and science?
    b. What instructional practices did the teachers utilize?

   c. What were their challenges adapting the design of the tools into their own instruction?

2. How did the students engage with computing and science in this learning environment?
   a. How did the students make connections between coding and science in this environment?
   b. What were their challenges engaging in learning to code and learning science through coding?

Chapter 2: Literature Review

In this chapter, I situate this study within a growing body of literature that focuses on introducing computational thinking into STEM classrooms (CT-in-STEM, Jona et al, 2014; Grover & Pea, 2017) and argue for the inclusion of a new approach into this research agenda - using physical computing devices for scientific inquiry. I begin this chapter by presenting the organizational framework for the literature review. Based on the review of literature, I operationalize key concepts used in this study, such as computational thinking and scientific inquiry, and construct the conceptual framework that guides the qualitative inquiry. I conclude this chapter with an additional review of the literature that guides the design of the computing tools and the assessment of computational thinking.

Organizational Framework

The goals of the literature review are: 1) to identify the gap between three intersecting bodies in the education literature, 2) to provide theoretical support from the literature for the integrated approach to computing and science advanced in this study, and 3) to construct the conceptual framework that guides the rest of the dissertation.

To achieve these goals, I organize my literature review with the framework presented in Figure 2. I identify three independent yet intersecting domains – computer science education, maker education, and science education, from which the conceptual framework of this study originates. I also pinpoint three lines of research that draw on knowledge from two of the three domains, which are presented as the three double-headed arrows. Within each domain, the review of the literature reveals a subfield

towards which scholarly discussions are gravitating, and I argue that the integrated approach of this study is situated at the point of convergence of these three fields.

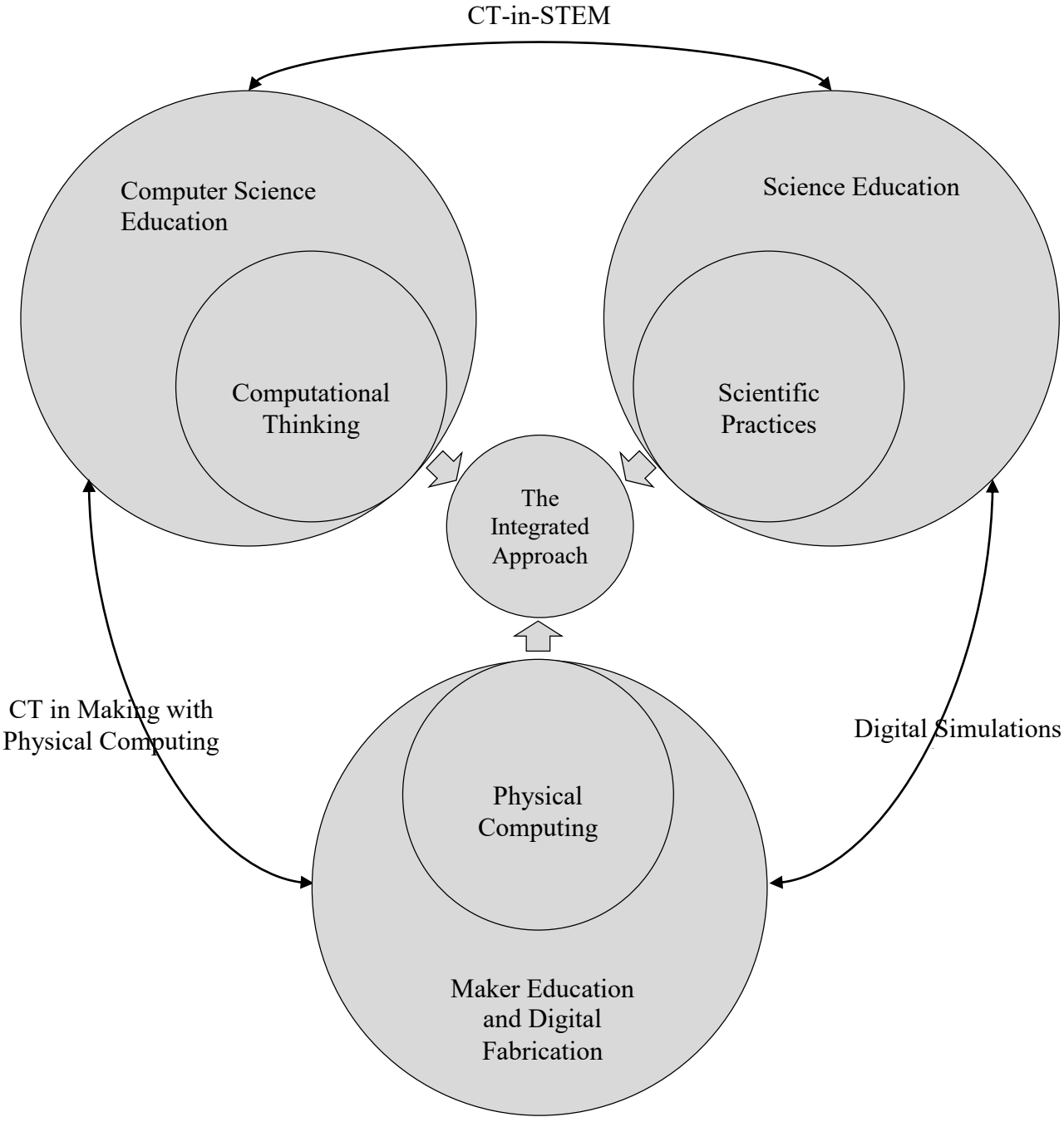

Figure 2. The organizational framework for the literature review.

The review of literature starts by presenting computational thinking as a universal competence as generally held by computer scientists (e.g. Wing, 2006, 2008, 2011; Aho, 2012) and computer science educators (e.g. Grover & Pea, 2013, 2017; Lee et al., 2013). The focus of the discussion with be the controversies about the definition (e.g. Denning, 2009) of computational thinking and difficulties in introducing it in computer science classes at the K-12 level (Jona et al., 2014). The review identifies an alternative perspective that addresses these difficulties - introducing computational thinking in STEM classes (Jona et al., 2014), especially some early promises achieved in science classrooms. This perspective also creates a research agenda that is largely to be fulfilled, and the integrated approach to computing and science belongs within this larger research agenda. Next, the review moves on to discuss the recent maker movement and the resulting popularity of digital fabrication labs (FabLabs; Blikstein, 2013), which supports the argument that physical computing adds a dimension of creativity and connectivity into the process of making and has potential for use in formal educational contexts. Finally, the review links the use of data with scientific inquiry, which has been a frequent topic in science education and make a case for physical computing devices as tools for scientific inquiry in science classrooms.

CS Education and Computational Thinking

Computational Thinking as a New Direction in CS Education

The term “computational thinking” was first coined by Jeanette Wing in 2006, who characterized computational thinking as a habit of mind of computer scientists who have a set of established ways to solve problems in computing, such as problem decomposition, recursion, parallelism, abstraction, and modularization. By arguing that

these ways of thinking actually stem from everyday life and enumerating real-life analogies of using these thinking patterns, Wing (2006) presented computational thinking as a "universally applicable attitude and skill set for everyone" (p.33). This idea is largely an extension of Seymour Papert's (1980) seminal work on teaching programming skills as a way of thinking for young children in that it separated computational thinking from programming. This notion catalyzed conversations among both computer scientists and computer science educators about what constitutes "computational thinking" and how to teach it to children.

Controversies in the Definition of Computational Thinking

Wing's notion of "to think like a computer scientist" also presents a somewhat vague picture of what computational thinking is. Wing later clarified her definition of computational thinking (2008) to include abstraction and automation as key components of computational thinking. Abstraction is a somewhat simplified representation of the real world and can be combined to represent the complexities of the real world. "Layers" (p.3718) of abstraction are critical in combining lower-level abstractions into higher-level abstractions to increase the level of complexities, and the automation of these abstractions achieves computing. In essence, Wing (2008, 2011) thought of computational thinking as the mental process of formulating computational solutions to real-world problems. Aho (2012) echoed this notion by defining computational thinking as "the thought process involved in formulating problems so their solutions can be represented as computational steps and algorithms" (p.833). He used "models" and "algorithms" in place of abstractions to further ground computational thinking in the field of computer science.

Skeptics of computational thinking, notably Peter Denning (e.g. 2007, 2009, 2017a, 2017b, 2017c), warned that this very notion might paint a narrow picture of the field of computer science. An author of the Great Principles of Computing framework, Denning (2011) argued that computing consists of computation, communication, coordination, recollection, automation, evaluation, and design, and that Wing's (2006, 2008, 2011) and Aho's (2012) definitions cannot adequately cover all those aspects (Denning, 2009). Neither is computational thinking insular or distinctive from the thinking of other disciplines, such as mathematical thinking, and design thinking. The idea that computational thinking might be beneficial for other subject areas could potentially be presumptuous because there is no empirical evidence that supports it (Denning, 2017a). Hemmendinger (2011) also argued that activities such as constructing models and recognizing patterns are not necessarily peculiar to computer science and thus not unique to computational thinking. Denning (2017b) advanced the term "computational design" (p.2) to avoid the fuzziness of both "computational thinking" and "computational doing" and more accurately capture the process of designing computational solutions to problems.

Disambiguation of Computational Thinking at K-12 Level

Despite the ongoing debate among computer scientists on the definition of computational thinking over a decade, computer science educators at the undergraduate and K-12 level embraced the concept with enthusiasm (Guzdial, 2008), as the term received attention from elected officials, tech companies, and funders (Wing, 2011). However, the lack of a clear, unified definition results in difficulty for instruction and assessment (Denning, 2017a). To mitigate this issue, computer science educators and

organizations came up with operational definitions of computational thinking appropriate for the level of students that they work with. Based on Wing's (2008) definition, Lee et al. (2011) gave a somewhat similar practical definition based on three pillars – abstraction, automation, and analysis, but their definition of abstraction involves simplifying real-world problems into bare minimal -- more practice-oriented than Wing's (2008) definition of abstraction. They also focused on the analysis or the thought process involved in validating the solution to a problem both in terms of correctness and efficiency. The operational definition of computational thinking from CSTA and ISTE (2011) characterizes computational thinking as the following: 1). formulating problems that can be solved computationally; 2) organizing and analyzing data; 3) representing data through models and other abstractions; 4) automating solutions; 5) improving solutions for efficiency, and 6) generalizing solutions and transferring them to other disciplines. Barr and Stephenson's (2011) definition encompassed even more elements (data collection, data analysis, data representation, problem decomposition, abstraction, algorithms and procedures, automation, parallelization, and simulation) and envisioned how these competencies would manifest themselves in classrooms of other STEM and non-STEM disciplines.

As more educators and practitioners came up with "practical" or "operational" definitions, the boundary of what constitutes computational thinking became ambiguous. Grover and Pea (2017) approached this issue by drawing a distinction between computational thinking concepts and practices. The former, which includes logical thinking, algorithmic thinking, pattern recognition, abstraction and generalization, evaluation, and automation, describes how computer scientists think, and the latter, which

includes problem decomposition, creating computational artifacts, testing and debugging, iterative refinement, and collaboration, illustrates what computer scientists do. Selby and Woollard (2013) reviewed definitions from computer scientists and educators alike and analyzed "core elements" shared across the definitions and "possible terms" only found in some definitions. They found that "a thought process," "abstraction," and "decomposition" were shared by most definitions, while "algorithmic thinking," "evaluation," and "generalization" were also well-defined concepts across disciplines. Kalelioglu, Gulbahar, & Kukul (2016) conducted a similar systematic review of the literature and reached a similar conclusion that problem solving and abstraction were the most common characteristics of computational thinking. They presented a framework that decomposed computational thinking as a process that has five stages – 1) problem identification, 2) data collection and analysis, 3) solution formulation, 4) solution implementation, and 5) solution evaluation. Relevant skills in computing, such as abstraction, debugging, and automation, can be used at each stage to solve the problem.

In the meantime, new perspectives of CT are still emerging. For example, Yaşar (2017) pointed out that defining CT as the habit of mind of domain experts is inherently problematic for instruction because it takes the accumulation of content knowledge and practical experience to engage the same thought process of experts. He proposed that addressing the "cognitive essence" of computational thinking, which appropriates how our brains store, retrieve, and process information could lead to more instructional activities. Although the empirical evidence cited in Yaşar (2017) sounds promising, this approach still needs to address the same issue of how to communicate clearly to teachers what computational thinking is.

Progress and Difficulties in Teaching Computational Thinking at K-12 Level

Despite the lack of consensus on the definition of computational thinking, the term sparked widespread interest among educators, education researchers, and policymakers (Grover & Pea, 2013). There have been independent efforts that surveyed the extensive literature on computational thinking in K-12 education. Most notable of these works are Grover and Pea's (2013) "state-of-the-field" review of computational thinking and K-12 and Mannila et al.'s (2014) multi-national survey focusing on the K-9 level. These researchers identified the following trends in the United States:

1. The focus of the field has shifted from disambiguating the definition of computational thinking to broadening access to computer programming and computational thinking in schools.
2. Multiple environments and tools have been developed to introduce computer programming to children and foster computational thinking, most notably graphical programming languages and environments such as Scratch (Maloney et al., 2011), Snap! (Harvey et al., 2014), App Inventor (Wolber, 2011), and Alice (Cooper et al., 2000) for digital story-telling and games, educational robots such as Bee-Bots, and combinations of these tools.
3. Multiple initiatives such as CS4All, CS4HS, CS Unplugged are increasingly pushing computing education into informal learning spaces in schools.

Lye and Koh (2014), in another extensive review of literature on learning of computational thinking through programming, also noted the proliferation of graphical programming languages and engaging activities such as digital-story telling and games. However, these researchers also identified the following issues remaining to be solved:

1. Although receiving increasing attention, issues of equity in CS have not been addressed adequately by programming education. Cooper, Grover, Guzdial, & Simon (2014) also concurred that computing education needs to be made available to females, underrepresented minorities, low-income students, and students with disabilities.
2. The application of computational thinking in other disciplines, especially STEM, is under-explored.
3. Teachers' understanding and engagement in computational thinking is another under-studied area.

Computational Thinking in STEM Education – A New Agenda

Perhaps the second issue above is the most problematic. If computational thinking is a general problem-solving competency, then programming should not be the only medium for it to develop. Lu and Fletcher (2009) argued that "[p]rogramming should not … be essential in the teaching of computational thinking, nor should knowledge of programming be necessary to proclaim literacy in basic computer science" (p.2). They suggested that students should not begin studying programming until sufficiently prepared to think computationally. In fact, both proponents and opponents of computational thinking agree that computing and computational thinking is deeply ingrained in other STEM fields (Wing, 2006, 2008; Denning, 2009) although they differ on whether computational thinking drives the advance of other STEM fields (Denning, 2007).

Jona et al. (2014) first formalized the concept of teaching computational thinking in STEM classrooms (CT in STEM, or CT-STEM). Citing the high demand for computer

science/STEM workers and the issue of underrepresented female and minority populations in computer science and STEM, they proposed that rather than introducing computer science as an independent discipline into schools, computational thinking can be embedded into STEM disciplines in such a way the former enriches the content of the latter in a context that is more established and accessible. This "symbiotic" (p.3) relationship reaches a broader combined audience and better prepares students for the challenges in modern STEM disciplines where computation is more ubiquitous (Wing, 2006; 2008). Grover and Pea (2013) hinted on this approach and later echoed this symbiotic relationship between computational thinking and STEM learning (2017). They called it a "CT-in-STEM-Learning" (p.33) agenda. Both groups of researchers independently pointed to the most pressing questions in this line of research, such as 1) What are the competencies in computational thinking that are most important for STEM disciplines? 2) What are the curricular activities that are conducive to computational thinking competencies in this context? and 3) How can we work with STEM teachers who can implement this approach?

It is worth noting that the notion of computational thinking in STEM learning actually predates the notion of computational thinking itself. Harel and Papert (1991) showed evidence that learning software design with the Logo programming language as a design tool led to a better understanding of fractions. More recent work studying computational thinking in STEM focuses on using a modern, agent-based descendent of Logo – NetLogo (Tisue & Wilensky, 2004) – for simulation and modeling in other STEM disciplines, including science (Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013; Wilensky, Brady, & Horn, 2014) and engineering (Blikstein & Wilensky, 2009).

Although all studies have shown promising results, they are examples of many ways computational thinking can be embedded into STEM learning, which warrants much more effort from both the computer science education and STEM education communities.

Maker Education: Digital Fabrication with Physical Computing Devices

One exciting way to embed computational thinking into STEM education is to investigate the use of physical computing devices in STEM classrooms. In their state-of-the-field review, Grover and Pea (2013) identified potentially powerful yet largely untapped spaces for computational thinking – makerspaces and FabLabs, which emerged in large succession following the maker movement. This section expands on Grover and Pea's (2013) argument and making a case for physical computing devices for computational thinking.

Maker Spaces and Digital FabLabs

Halverson and Sheridan (2015) defined "the maker movement" informally as the growing communities of people engaged in the creative process of producing artifacts in everyday life and sharing their processes and products with each other on physical and digital platforms. Dougherty (2012) distinguished "makers" from "inventors" or "tinkerers," to highlight the highly informal and creative nature of "making." Martin (2015) characterized the "maker mindset" as playful, asset- and growth-oriented, failure-positive, and collaborative. Also emerging in large numbers are physical spaces for making such as makerspaces, which are increasingly seen in after-school spaces, libraries, and museums. Regional maker fairs are held internationally that attract large crowds. Gershenfield and colleagues (Gershenfeld, 2012, Mikhak et al., 2002) created the first FabLab at MIT with low-cost physical computing devices for individuals to engage

in digital fabrication, which is now spreading across the globe (Blikstein, 2013). The maker culture has grown so rapidly that it caught the attention of policymakers and political figures, including President Obama (White House, 2014).

The promise of the maker movement for education has caught the attention of educational researchers who saw the deep theoretical roots of making. Halverson and Sheridan connected making to Deweyian constructivism of learning by experimenting, playing, and authentic inquiry (Halverson & Sheridan, 2014). Martin (2015) suggested that experimentation with ideas allows learners to check expectations against reality which could lead to conceptual disequilibrium and eventually conceptual adaptation in Piagetian learning theories (Piaget, 1950) while the social and shared nature of making is one important component in social constructivism (Vygotsky, 1978). Blikstein (2013) and Martinez and Stager credited Papert's work on constructionism for laying the theoretical foundation for the maker movement. Constructionism (Harel & Papert, 1991) holds that the creation of knowledge happens when students create and publicly share objects. Thus, it distinguishes itself from the Deweyian, Piagetian, and Vygotskian constructivism by emphasizing both the hands-on and social nature of making.

Physical Computing Tools for Making

One important characteristic of the maker movement is the availability of low-cost digital tools readily available for digital fabrication (Martin, 2015; Blikstein, 2013; Stager, 2013). These tools include "digital physical tools" such as 3D printers, laser cutters, and digital embroidery machines that allow novices to create artifacts that are high both in aesthetics and quality and "digital logic tools" such as low-cost, hobbyist friendly physical computing devices including microcontrollers to control external

devices such as sensors, motors, and switches (Martin, 2015). Familiarity with these tools moves students from discrete skills to digital and computing literacy (Blikstein, 2013; Blikstein & Krannich, 2013). They also provide students with learning opportunities to break down technologies and see how they work. This approach adds transparency and marks a shift away from otherwise "black box" models of technology where youths know how to work with technology but do not understand how it works (Resnick, Berg, & Eisenberg, 2000). The FabLabs also provide an opportunity for educators to design collaborative learning environments that are student-centered, competence- and success-focused, and failure-safe (Stager, 2013).

Obstacles to Making in Education

The maker movement is also facing obstacles. First, educational makers need to tackle the issue of access and equity. Although much attention has been devoted to broadening the access of making to girls and under-represented minorities (e.g. Brady et al. 2016), the maker movement needs to break away from its "white male nerd" stereotype and serve more individuals from under-represented groups (Halverson & Sheridan, 2014). Second, the maker movement needs to align making with content standards and make more connections between making and formal education (Martin, 2015). Finally, some of the tools used in maker education such as the Arduino are very high entry thresholds (Hughes Gadanidis, & Yiu., 2016), and more digital tools should be investigated or developed to lower the cognitive demand on young and novice makers.

The Future of Making in Formal Education Spaces

Although the promise of making and the maker movement themselves deserve much scholarly attention on learning in informal makerspaces and FabLabs, educational

researchers are also excited about the possibility of introducing making into formal educational spaces and using making as a venue to teach the content of other disciplines. For example, Brady et al. (2016) modeled the future of computing and computer science education after making and maker activities, noting that the use of low-cost embedded devices provides pathways for more inclusive computing education. They investigated the use of wearable connected devices to create contexts for computing that are more friendly to girls. Berry et al. (2010) explored the possibility of using digital fabrication tools to infuse engineering design principles in elementary mathematics classrooms. Since computing is heavily involved in making, the implication of making for fostering computational thinking is being investigated. Rode et al. (2015) advocated the move from computational thinking to computational making, arguing that components of computational thinking can be embedded into activities of making, such as data collection and analysis, identifying and implementing, and testing solutions, modeling data and simulations, decomposing problems and automating solutions. Kotsopoulos et al. (2017) also included making in their pedagogical framework for computational thinking by highlighting digital making as one of their four pedagogical experiences that foster computational thinking – unplugged, tinkering, making, and remixing.

Physical Computing Devices and Computational Thinking

Rode et al. (2015) argued that analyzing and logically organizing data is an essential step in using computational thinking in making. If that is the case, physical computing devices' ability to automatically collect, organize, and analyze data can be instrumental in developing computational thinking skills. Work in this respect is emerging. Brady et al. (2017) designed the Wearing the Web activities that use user-

programmable badges (connected microcontrollers). The learners were introduced to core computational science ideas such as abstraction, data representation, and dimensions of human-computer interaction (HCI) with physical activities that use these devices to simulate the structure of real computer networks. Dasgupta and Resnick (2014) noted that while understanding and using data are becoming increasingly important skills in our lives, introductory courses in computer science usually focus more on the process of programming and rarely go beyond the simple use of data such as creating variables, using lists and key-value pairs. Physical computing devices such as sensors offer a viable venue to expose learners to more meaningful use of data. Although such literature is emerging and limited to the context of computer science education, the ability of physical computing devices to automatically collect, organize, and analyze data might also prove useful in science classrooms, which is the focus of the next section.

Science Education and Scientific Practices

Need for Scientific Practices in Science Classrooms

A quote allegedly from Seymour Papert observes that a teacher from the 16th Century would have no problem teaching a class today (Blikstein, 2013). This is especially true in science. Dewey (1910) criticized the excessive focus in science instruction on facts and the lack of focus on teaching thinking and an attitude of mind. Schwab (1958) noted the "vast increased rate at which data are nowadays accumulated and processed" and suggested a change in the teaching of science as inquiry, which corresponds to the changes in science as a field that acquired a "dynamic outlook" (p.374). However, in the 21st Century, inquiry is still not part of the science curriculum (Etheredge & Rudnitsky, 2003; Wilcox et al., 2015). Hanauer et al. (2006) observed that

"the goals of scientific research and current pedagogical practices are at odds" and that "the focus on persuading students of the correctness of stated information" and "the ensuing culture of conformity with established knowledge is the very antithesis of scientific inquiry" (p.1880).

The absence of scientific inquiry in K-12 can largely be attributed to the confusion among K-12 teachers of science about its definition and the varying interpretations by teacher educators of science (Barrow, 2017; Demir & Abell, 2010). There are also differences between laboratory settings and classroom settings that render science classrooms suboptimal as venues for scientific inquiry (Hanauer, 2016). To clear some of this confusion in the science education community about inquiry among the science education community, the Next Generation Science Standards (NGSS Lead States, 2013) employed the term "scientific and engineering practices" to refer to the activities in which scientists are engaged to build models and understand the world and those in which engineers are engaged to design and build systems[1]. The standards emphasized scientific practices: "Students cannot comprehend scientific practices, nor fully appreciate the nature of scientific knowledge itself, without directly experiencing those practices for themselves" (p.xv).

The use of mathematical and computational thinking is included as one of the science and engineering practices in NGSS (NGSS Lead States, 2013). The National Research Council (NRC, 2012) recommends the use of computing tools in scientific practices, suggesting that "computational methods are … potent tools for visually

[1] To align with the standards, this study will adopt the term "scientific practices" henceforth. However, studies that predate NGSS still used the term "scientific inquiry," and this study will maintain the consistency of the language with the original literature.

representing data, and they can show the results of calculations or simulations in ways that allow explorations of patterns" (p.65). This, however, is a somewhat narrow and even erroneous perspective of what computational thinking is. Wilkerson and Fenwick (2018) presented a much-balanced view, suggesting that students who are engaged in computational thinking decompose problems and use computer tools and algorithms to automate jobs such as data collection, analysis, or testing theories. Using computer tools for simulations is one way of using computational thinking, yet it is also about building tools to answer questions. Sneider, Stephenson, Schafer, and Flick (2014a, 2014b) also concurred that the use of computer simulation could enhance understanding of phenomena that students cannot normally experience in person. They added that computing tools could also automated collection and analysis of data and data mining, the practice of analyzing existing large-scale data to extract insights.

As NGSS practices are interconnected, the practice of using computational thinking also engage students in analyzing and interpreting empirical data, which they use in developing and using models (Wilkerson & Fenwick, 2018), building arguments with evidence (McNeill & Berland, 2017, Berland, McNeill, Pelletier & Krajcik, 2016), and constructing scientific explanations (McNeill & Krajcik, 2008; McNeill, Berland & Pelletier, 2017). The collection, analysis, and interpretation of data is also a computational thinking practice (CSTA & ISTE, 2011; Grover & Pea, 2017). In the meantime, data are also a fundamental form of abstraction in computational thinking and the building blocks of layers of abstraction (Wing, 2006; 2008). Therefore, learning activities that leverage computing tools to collect and analyze data could potentially be another approach of embedding computational thinking into science classrooms.

Physical Computing Tools: New Possibilities for Scientific Practices

Physical computing devices, with their small footprints and abilities to collect data automatically (Dasgupta & Resnick, 2014), potentially make good tools for these learning activities in science classrooms. The use of physical computing tools for scientific practices has captured the attention of researchers although the exploration is still at an early stage. Song (2014) described a one-year "Bring Your Own Devices (BYOD) for seamless scientific inquiry" project where students examine the use of mobile devices for ubiquitous data collection beyond the classroom. He showed positive learning outcomes as well as positive attitudes towards ubiquitous inquiry with mobile devices. Davis (2017), arguing that using physical computing projects for automated data collection offers "a unique opportunity to engage in inquiry and creation that is grounded in collecting, analyzing, and communicating data" (p.84), presented a "Talking Window Garden" project where elementary school students built plant pots and made them smart with sensors to collect and data about their plants. These two research studies, especially the latter, demonstrate the connection between physical computing devices for scientific practices in terms of automated data collection and analysis, but no connection to computational thinking was made. Nevertheless, a recipe for designing physical computing devices for science is present across the emerging literature: affordable hardware that can be easily programmed with an educational programming language that facilitates data collection, connectivity between the device and the Internet, an automatic data visualization platform, room for design for students, and opportunities to ask questions about data.

Summary

The previous sections review the literature and connect three seemingly separate areas of research in computing education, maker education, and science education to argue for an interdisciplinary approach to solve the difficulties in these areas. The review shows that although high-level debates over the definition of computational thinking continue today, at the K-12 level, computer science educators have come to agree upon a stable and operationalized definition of computational thinking (Grover & Pea, 2017) that incorporates the essential concepts and practices to practitioners. While maker spaces and digital fabrication labs provide informal spaces for using technologies such as microcontrollers to develop computational thinking, it is possible to use computing devices to create a data-rich learning environment in K-12 science classrooms that employ scientific practices, and emerging literature is beginning to explore this possibility.

Conceptual Framework

Based on the synthesis of the literature, I advance the following conceptual framework (Figure 3). The conceptual framework extends the literature review and presents a more accurate view of the integrated approach of computing and science on which this study is based, with key concepts such as computational thinking and scientific inquiry operationalized as follows:

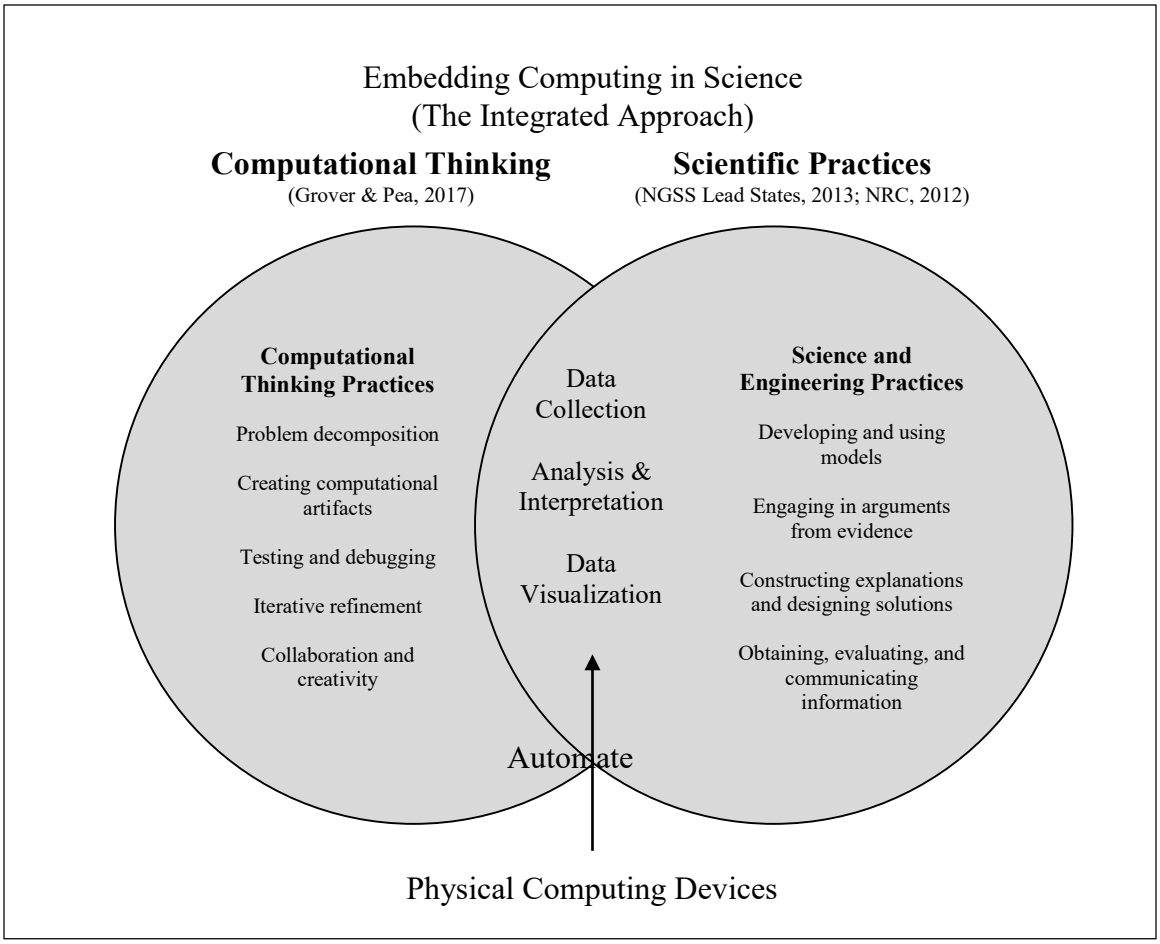

Figure 3. The conceptual framework used in this study.

Computing: this study approaches computing by focusing on developing computational thinking, a universal skill for everyone. Despite the ongoing debates among computer scientists (e.g. Denning, 2017a, 201c) over a formal definition of computational thinking, the operational definitions for educators and practitioners are showing trends of converging. This study adopts Grover and Pea's (2017) dichotomy of computational thinking "concepts and practices" as the operationalized definition. This framework (Table 1) echoes the conclusions of comprehensive literature reviews on definitions of computational thinking (Selby & Woolard, 2013; Kalelioglu, et al., 2016) and distinguishes between what is latent (computational thinking concepts) and what is

observable (computational thinking practices), providing support for qualitative data analysis in later chapters.

Table 1

Computational Thinking Concepts vs. Practices.

Item	Definition
Computational Thinking Concepts:	
• Logic and logical thinking:	Analyzing situations to reach a reasonable conclusion or decision. In computer science, this specifically refers to the use of Boolean expressions: and, or, not, and combining if conditionals with them.
• Algorithms and algorithmic thinking:	Thinking and planning step-by-step. In computer science, this usually involves sequence, selection, and repetition (loops).
• Pattern and pattern recognition:	Reaching generalizable solutions from identifying patterns
• Abstraction and generalization:	A core idea of computer science involving representing real-world entities. Data is a fundamental form of abstraction.
• Evaluation:	Evaluating solutions to problems not only of correctness but also effectiveness.
• Automation:	Working towards a solution that can be carried out by a machine. Also, being able to differentiate what problems are better solved by a human or a machine.
Computational Thinking Practices:	
• Problem decomposition:	Breaking complex problems into smaller, more manageable components.
• Creating computing artifacts:	Creating computing software and/or hardware to solve problems.
• Testing and debugging:	Evaluating the accuracy of a solution and being able to fix problems when they occur.
• Iterative refinement:	Having a mindset for iteratively improves the solution to problems.
• Collaboration and creativity:	Collaborating with others and using out-of-box thinking to solve problems.

Scientific practices: this study approaches science by focusing on scientific and engineering practices that NGSS (NGSS Lead States, 2013) adopted. These are interconnected practices, including: analyzing and interpreting data, using mathematical and computational thinking, developing and using models, engaging in arguments from evidence, and constructing explanations and designing solutions.

Physical Computing Tools: Following these operationalized definitions, I place data analysis and interpretation at the intersection of computational practices and science and engineering practices. Physical computing devices provide the capabilities of supporting data collection with sensors and data visualization with easy-to-use data dashboards for analysis and interpretation. In designing, building, and programming computing devices, students are engaged in other computational practices, such as creating computational artifacts, testing and debugging, and iterative refinement. By interpreting data, students are also engaged in other science and engineering practices, such as developing models, constructing explanations, and using arguments from evidence.

Methodological Considerations

This conceptual framework provides a foundation for the use of physical computing devices as tools for embedding computational thinking in scientific inquiry activities. However, to my knowledge, there are few computing tools designed specifically for this approach, which highlights the need to design a set of computing tools that supports the implementation of this approach. This section focuses on reviewing additional literature on frameworks that guide the design of such tools.

Design Frameworks of Educational Software for Scientific Practices

The lack of connection between physical computing devices and computational thinking prompts the need to design computing tools that explicitly make this connection. Three frameworks have been developed by educational technology researchers and learning scientists that provide guidelines for developing computing tools and software that support computational thinking and scientific practices.

The first framework by Repenning, Webb, and Ioannidou (2010) is the broadest among the three and focuses on properties of software compatible with public K-12 schools and conducive to computational thinking. They use the notion of computational thinking tools to refer to a combination of computational thinking curriculum, authoring tools, and teacher training. Such tools should meet all these requirements:

1. **Low threshold:** novices can learn to command these tools quickly.
2. **High ceiling:** learners should be able to use these tools to create complex projects.
3. **Scaffold flow:** these tools should provide stepping stones at different stages with plenty of scaffolds and challenges.
4. **Enable transfer:** skills learned with these tools can be applied to other applications.
5. **Support equity:** these tools should be equally accessible and motivational to all learners.
6. **Systemic and sustainable:** all teachers can use these tools to teach all students, i.e., support teacher training and standard alignment.

Although this framework is not specific to software tools used for other disciplines, the "enabling transfer" component does highlight the importance of skills learned being transferred to other contexts. It is broad in scope in that it does not prescribe what types of computational thinking or activities in which the computational thinking tools should engage the learners. This aspect is complemented by the second framework by Windchitl (2000). This framework proposes a set of methods of science and inquiry that software should support. These methods include visually enhanced data

analysis, simulations and microworlds, and modeling. While Windchitl's work was published in 2000, and the software tools mentioned cannot compare with their modern counterparts, it is remarkable that most work on CT-in-STEM focuses on the methods of science that he mentioned. He also pointed out the issues of software-supported inquiry environments. First, learners do not necessarily engage with these tools in meaningful and productive ways. Second, software-supported inquiry environments provide learners with simplified versions of reality and do not engage them with modes of inquiry other than quantitative, such as qualitative, philosophical, and personal. Third, the use of technology needs to be organically integrated with classroom instruction.

The third framework (Quintana et al., 2004) comes from learning scientists and focuses on software scaffolding design. Using a theory-driven approach, the authors identified three components of inquiry – sense-making, process management, and reflection, and articulation. Then, based on the obstacles learners might face when engaging in these cognitive processes of inquiry, they offered guidelines and strategies for software-supported scaffolding. In supporting sense-making, they recommended the use of representations and language understandable to learners, tools and artifacts around the disciplinary semantics, and representation that learners can inspect in various ways. In supporting process management, they suggested that scaffolding should provide structure for complex tasks, embed guidance from experts, and automation of less important tasks. In supporting articulation and reflection, they advocate designs that promote discussion and reflection during the investigation. Figure 4 details the strategies that the authors recommended.

Scaffolding Guidelines	*Scaffolding Strategies*
Science inquiry component: Sense making	
Guideline 1: Use representations and language that bridge learners' understanding	1a: Provide visual conceptual organizers to give access to functionality
	1b: Use descriptions of complex concepts that build on learners' intuitive ideas
	1c: Embed expert guidance to help learners use and apply science content
Guideline 2: Organize tools and artifacts around the semantics of the discipline	2a: Make disciplinary strategies explicit in learners' interactions with the tool
	2b: Make disciplinary strategies explicit in the artifacts learners create
Guideline 3: Use representations that learners can inspect in different ways to reveal important properties of underlying data	3a: Provide representations that can be inspected to reveal underlying properties of data
	3b: Enable learners to inspect multiple views of the same object or data
	3c: Give learners "malleable representations" that allow them to directly manipulate representations
Science inquiry component: Process management	
Guideline 4: Provide structure for complex tasks and functionality	4a: Restrict a complex task by setting useful boundaries for learners
	4b: Describe complex tasks by using ordered and unordered task decompositions
	4c: Constrain the space of activities by using functional modes
Guideline 5: Embed expert guidance about scientific practices	5a: Embed expert guidance to clarify characteristics of scientific practices
	5b: Embed expert guidance to indicate the rationales for scientific practices
Guideline 6: Automatically handle nonsalient, routine tasks	6a: Automate nonsalient portions of tasks to reduce cognitive demands
	6b: Facilitate the organization of work products
	6c: Facilitate navigation among tools and activities
Science inquiry component: Articulation and reflection	
Guideline 7: Facilitate ongoing articulation and reflection during the investigation	7a: Provide reminders and guidance to facilitate productive planning
	7b: Provide reminders and guidance to facilitate productive monitoring
	7c: Provide reminders and guidance to facilitate articulation during sense-making
	7d: Highlight epistemic features of scientific practices and products

Figure 4. Summary of the scaffolding design framework. Adapted from Quintana, C., Reiser, B. J., Davis, E. A., Krajcik, J., Fretz, E., Duncan, R. G., … Soloway, E. (2004). A Scaffolding design framework for foftware to support science inquiry. *Journal of the learning sciences.*

Summary

The review of literature has placed this study's approach to computational thinking at the intersection of three established bodies of literature: teaching

computational thinking, making and digital fabrication, and teaching science as inquiry. Although little research has been done using this approach *per se*, there is evidence of theoretical support in previous research for the use of physical computing devices for inquiry activities, although the design of the computing tools will be key. The review of the literature has identified three design frameworks at different levels that guide the design of the computing tools. The next chapter focuses on how the design of the set of computing tools and curriculum fits into the three frameworks.

Chapter 3: The Design of the Computing Tools

Previous chapters have laid theoretical and empirical support for the integrated approach of computing and science. In this chapter, I will describe the design of the computing tools used in the Smart Greenhouse project that put the integrated approach in practice. These tools supported 200 8th Graders and their teachers to build smart greenhouses within three weeks. I will first give an overview of the components of the tools. Afterward, the discussion of the design will center on the decision to use the MicroPython TPL instead of some variant of a BPL. The focus will be on the general principles under which the TPLs are extended for use with students and teachers who are complete novices. The discussion will also cover the design decisions on the selection of the computing hardware and the ideas that undergirded the design of the lessons based on the computing tools.

Overview of the Tools

The "computing tools" referred to henceforth are comprised of 1) the GrowThings library that extends the MicroPython TPL, 2) the Wio-Link microcontroller board on which MicroPython operates, and 3) a selection of 11 "Grove" devices that have a compatible physical interface with the Wio-Link board. Additionally, a series of lessons that focus on familiarizing users with these tools and using these tools to learn science were also developed. These components form a set of open-source and low-cost computing tools that middle school science teachers and students can use to prototype ideas quickly and build complex computing artifacts. Thanks to the MicroPython TPL for microcontrollers and the GrowThings library designed to provide an accessible interface

to that language, novices can start programming the microcontroller boards to communicate with the Grove devices within minutes.

TPLs vs. BPLs

Central to all design decisions is the selection of (Micro)Python as the interface between the users and the microcontrollers in this project. Block-based programming languages (BPLs) such as Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010), Alice (Cooper et al., 2000), Blockly (Trower, & Gray, 2015), and App Inventor (Wolber, 2011) currently dominate the space of computing education in primary and early secondary schools (Kolling, Brown, & Altamri, 2015; Tsukamoto et al., 2015). However, out of the education space, text-based programming languages (TPLs) are almost exclusively used in the industry. BPLs are comparable to training wheels: they lower the threshold for complete novices to experience programming and learn fundamental programming concepts, but once they become proficient in BPLs, they should transition to TPLs. Otherwise, BPLs will become a hindrance. Brian Kernigan, a trailblazer in computer science, compared learning to program with BPLs as climbing trees to go to the moon: the support that BPLs provide will eventually become limiting.

However, unlike cycling with and without training wheels, coding with BPLs and TPLs are different experiences, and transition from BPLs to TPLs requires closing a much wider gap. Researchers are starting to investigate ways to mitigate the “ceiling effect” of BPLs, and some (e.g. Tabet, Gedawy, Alshikhabobakr, & Razak, 2016; Kolling, Brown, & Altadmri, 2015, Robinson, 2016) have proposed tools and instructional techniques that have yet gained traction in the computing education community. Harvey and Monig (2010) suggested keeping the training wheels, arguing

that BPLs such as Scratch may be used to introduce advanced concepts in programming. However, they did not discuss how to use Scratch beyond the context of teaching and learning instruction.

Just as not everyone learns to cycle with training wheels, coding does not have to begin with BPLs. The design of BPLs is strongly influenced by Papert's (1986) Logo programming language. However, the Logo language emerged when barriers to TPLs were high (Grover & Basu, 2017). Literature from the 1980s and 1990s also highlighted that understanding advanced constructs of programming languages was challenging and frustrating for beginners (e.g. Du Boulay, 1986; Mayer, 1989; Ebrahimi, 1994). However, this body of literature largely predates modern programming languages such as Python, an open-source, general-purpose TPL designed to be friendly, robust, and versatile for absolute novices (van Rossum, 1999). Leading undergraduate computer science programs have adopted Python as their instructional language for introductory computer science courses (Shein, 2015). Researchers investigating whether students from lower age groups can use Python found positive results in high school (Grandell, Peltomaki, Back & Salakoski, 2006) and elementary schools (Tsukamoto et al., 2015). While there is very little empirical evidence that supports that either Python or BPLs produces better learning outcomes, learning Python at the outset eliminates the need for transition: Python is already a popular choice in computational science (Langtangen, 2006, Carver, Chue Hong, & Thiruvathukal, 2017), artificial intelligence, and data science. Therefore, Python embodies the properties of low threshold and high ceiling (Repenning et al. 2013), which makes it a better fit for this context.

Multiple characteristics make Python a beginner-friendly language. First, it has a small, consistent, and expressive syntax – there are not many syntax rules to memorize and not many of the idiosyncrasies of other programming languages. The absence of symbols such as curly brackets ({}) and semicolons (;) seen in many other programming languages results in clean and readable code that is less intimidating to absolute beginners. Second, Python is a dynamically typed programming language. It does not require the user to specify the type of variables, an intimidating concept to beginners and better suited for a college-level computer science class. Third, Python is a high-level programming language that automatically handles control of hardware so that learners do not need to worry about freeing up memories after certain variables become redundant. Experienced users can also easily extend the language by installing or creating third-party packages widely available in Python ecosystems to handle virtually every task imaginable. Fourth, Python is an interpreted language that allows scripting, the preferred mode of programming in scientific computing (Longtangen, 2006) in line with other popular scientific computation software packages such as Matlab, S-Plus/R, and Mathematica.

With Python's extensibility, it is possible to extend Python by incorporating some of the design principles of BPLs to make it even more friendly to beginners. BPLs are considered to be better suited for computing instruction in lower-age groups because they provide interactive environments that support creation of media-rich projects (Maloney et al, 2010, Maloney, Peppler, Kafai, Resnick, & Rusk, 2008), such as digital storytelling, games, and science projects, which add to the joy of computing (Harvey & Monig, 2010; Tsukamoto et al., 2015). In other words, BPLs provide content-rich contexts for

programming. TPLs, while unable to replicate the beginner-friendly drag-and-drop interface of BPLs, can undoubtedly be improved by borrowing this design principle of BPLs and adding mechanisms that support content-rich projects such as building smart greenhouses. The "GrowThings" library was created for this purpose. The next section will describe the five principles that supported the design of the GrowThings library.

The Five Principles of Extending TPLs

The GrowThings Library

The GrowThings library extends the base MicroPython TPL and adds a set of functionalities to the language to make it more accessible for beginners. The MicroPython language (George, 2017) is a subset of the Python language designed to run on microcontrollers. It inherits the same syntax from Python and is thus indistinguishable from its full-fledged sibling in the eyes of novices. Therefore, henceforth this study will use Python and MicroPython somewhat interchangeably. MicroPython is developed mainly for hobbyists and the DIY community and assumes some familiarity with electronics. Therefore, the language without any modification is riddled with jargon. The left column of Table 2 demonstrates some of the code that the users would have had to write if using only base MicroPython. Jargons such as TSL2561 (the model name of the light sensor) and I2C (a data communication protocol), PWM, SDA, and SCL pins obscure the meaning of the code. Programming each device would require familiarity with how it communicates with the microcontroller differently from other devices. Consequently, what one learns from programming one device might not apply to the programming of another device of the same type.

In order to bypass these technical jargons and make learning transfer possible, it is necessary to provide a set of new, unified, and simple programming interfaces, known as APIs (Application Programming Interfaces) so that absolute beginners can make sense of the code and learn to program with MicroPython quickly. The idea is to wrap the complex code in the left column of Table 2 into functions with more meaningful names. The right column of the same table shows code snippets that achieve the same purpose as the ones on the left but are written with the GrowThings library. They are less jargon-ridden, more succinct, and much easier to read. More importantly, one's knowledge in programing one device can become the basis on which one reasons about programming another. These improvements are essential for maintaining a low threshold for beginners to learn a new language quickly so that they can apply their programming skills to other learning tasks.

Table 2

A comparison of code written in base MicroPython vs. GrowThings.

Base MicroPython	**With GrowThings Library**
Read temperature from a temperature sensor:	

`from dht import DHT22`
`from machine import Pin`
 	
`d = DHT22(pin=Pin(5))`
`d.measure()`
`d.temperature()`
`d.humidity()`
	`from sensors import TemperatureSensor`
	`ts = TemperatureSensor(port=3)`
	`ts.get_temperature()`
	`ts.get_humidity()`

Read lux value from a light sensor:	

```
from tsl2561 import TSL2561
from machine import I2C

i2c = I2C(scl=5, sda=4)
tsl = TSL2561(i2c, address=0x29)

tsl.activate()
tsl.read()
```

```
from sensors import LightSensor
ls = LightSensor(port=6)
ls.get_lux()
```

Rotate a servo:

```
from machine import PWM, Pin
pwm = PWM(Pin(5), freq=500)
pwm.duty(122)
pwm.duty(60)
```

```
from actuators import Servo
s = Servo(port=1)
servo.set_position(degree=90)
servo.set_position(degree=0)
```

Make the LED strip blink 3 times in 1-second intervals:

```
from neopixel import NeoPixel
from machine import Pin
import time

np = NeoPixel(Pin(4), 30)

for i in range(3):
 for j in range(30):
 np[j] = [255, 255, 255]
 np.write()
 time.sleep(1)
 for j in range(30):
 np[j] = [0, 0, 0]
 np.write()
 time.sleep(1)
```

```
from displays import GrowLight
gl = GrowLight(port=2)
gl.blink(color=[255, 0, 0], times=3,
interval=1)
```

The design purposes of the GrowThings library are 1) to minimize the complexity of programming computing devices with MicroPython; 2) to maximize the compatibility of Python with the science instruction in a middle school classroom; and 3) to align the experience of coding with the GrowThings library with that of programming with other real-world Python libraries. To achieve these purposes, I developed five principles that guided the design of the GrowThings library. These five principles not only incorporated the design principles of BPLs but also drew lessons from the design of widely-used

applications in computing education such as Scratch, the SenseHAT library, the CodeCombat website (http://codecombat.com), the RoboSim educational robots (Gucwa & Cheng, 2014), Apple's Swift Playground on iPad, and the turtle library that comes with Python. Adhering to principles in object-oriented software design (Goldwasser & Letscher, 2008), the principles also drew inspiration from popular Python libraries used in the industry, such as the Scikit-Learn package (Buitinck et al., 2013) for general-purpose machine learning and the Keras package (Chollet, 2017) for rapid development of deep neural networks for artificial intelligence. The next sections will describe each of these five principles.

1. The Modularity Principle

In software design, it is essential to design APIs that are organized intuitively and logically, so that the users can reason about the structure of the APIs (Goldwasser & Letscher, 2008). This idea is analogous to filing paper documents within cardboard file folders. Most BPLs also organize the blocks into categories so that they are easier to find. The current version of the GrowThings library provides APIs for 11 devices essential to the automation of the smart greenhouses. These devices include "sensors," devices that turn environmental information into electronic signals, "actuators," devices that move physically, and "displays," devices that display information. The library categorized APIs using the same taxonomy for users to locate them conveniently[2]. For example, in order to access the APIs associated with the temperature sensor, a user, knowing that the

[2] This taxonomy is problematic and a future version of the library has adopted a new taxonomy. Please see Chapter 8 for more details.

temperature sensor is of the "sensor" type, can locate functionalities related to the temperature sensor by writing the following code:

```
from sensors import TemperatureSensor
```

Likewise, functionalities related to relays, electronically controlled switches, can be accessed with:

```
from actuators import Relay
```

In the code above, "from" and "import" are reserved words Python knows to have specific meanings, i.e., to locate specific groups of APIs. "sensors" and "actuators" are the abstract "folders" into which Python looks to access functionalities of devices. "TemperatureSensor" and "Relay" are names of classes. Classes are the building blocks of object-oriented programming (OOP) and software design that allow programmers to create abstract representations of objects in the real world, mimicking their properties and behavior. For example, cars as a class in the real world share common properties such as having wheels and behave in similar ways such as moving forward and backward. Classes in OOP seeks to logically organize representations that have similar properties and behavior in similar ways to achieve better organization and encourage code reuse. As a result, the behavior of temperature sensors such as measuring temperatures is all organized within TemperatureSensor class for the users to access. Figure 5 demonstrates the class hierarchy of the GrowThings library.

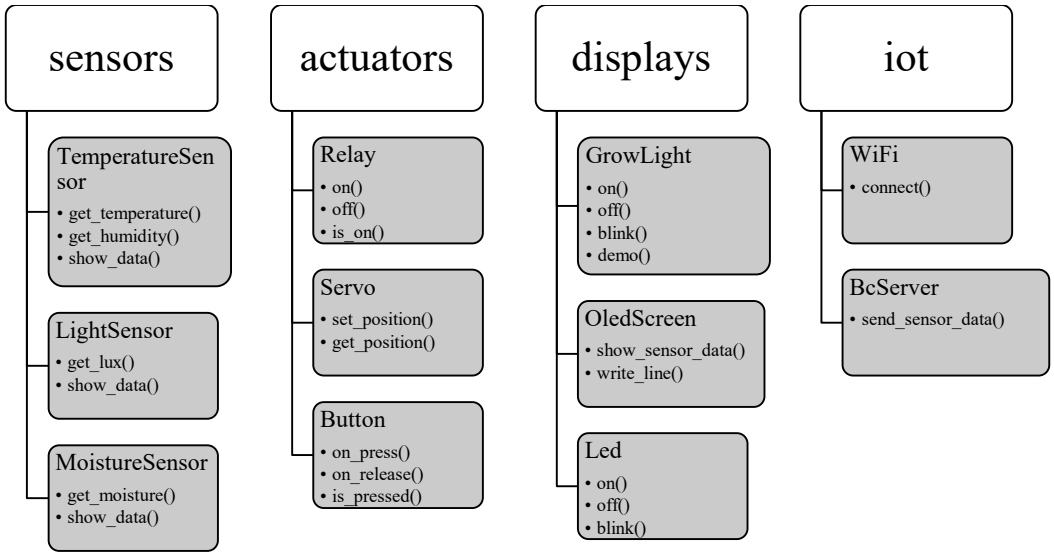

Figure 5. The structure of the GrowThings library.

Similar to the reasoning that classes that behave differently need to be put into different "folders," it is also possible to reason that the classes in the same "folder" should also share similar behavior. In this case, all classes in the sensors "folder" can measure certain variables and therefore have "get" methods for users to obtain readings of certain variables such as temperature and light intensity (lux). They are also capable of showing the data on an OLED screen. Hence the "show data()" function. This design enables users to reason how other classes within the same "folder" should behave so that once they become familiar with one class, they can transfer their knowledge to working with other classes of similar types.

This class hierarchy allows entities in it to be named consistently following widely-accepted conventions to avoid mistakes in spelling and capitalization. Python is a case-sensitive programming language that will not forgive the slightest inaccuracies in spelling and capitalization, which is a common struggle for beginners and experienced coders alike. To make matters worse, the language itself does not enforce any

conventions or rules, and as a result, “TemperatureSensor,” “temperature sensor,” and “Temperature_Sensor” are all valid names in Python. The GrowThings library is designed to reduce the chances of making mistakes in capitalization and spelling by enforcing strict naming rules, depending on where the entity falls on the class hierarchy. Names of modules, or “folders,” at the top of the hierarchy are in lower case and end with a plural “s.” The next level below, classes, follows the “CamalCase” convention, which constructs names consisting of multiple words by capitalizing the first letter and joining each word with no space or symbols. All functions enclosed in classes known as methods are in lower case with words joined by underscores (“_”). These rules come from general Python naming conventions and thus are also present in other Python packages that users might also encounter. Consistent APIs within both the library and the ecosystem decrease the users’ time to learn to use the APIs and minimizes their efforts when they switch to other programs. They reduce the chance for errors and improves the readability of the code beginners write.

2. The Semantic Transparency Principle

One of the most significant advantages BPLs have over TPLs is semantic transparency. The blocks of BPLs are typically labeled with their exact functions in the plain human language, e.g. “move the cat left by 200 pixels,” or “measure the air temperature.” TPLs, on the other hand, are typically designed to issue instructions or commands to the machines. Programmers often need to perform mental translation from what they want the computer to do to a set of instructions to the computer to achieve that purpose. For example, in order to turn on an LED, the programmer needs to instruct the microcontroller to set the pin to which the LED is connected to a high output level.

"Turning on the LED" is the desired action, and "setting output level on a pin" is the instruction.

A second issue specific to the MicroPython TPL in this context is that existing MicroPython code tends to use the model name of the computing devices rather than their functions as the class names. For example, the temperature sensor used in the smart greenhouse project is known as "DHT11," and MicroPython offers a corresponding "DHT11" class to control it. However, in this context, names such as "DHT11" make much less sense to students and teachers in a science classroom than names such as the "temperature sensor" and might result in steeper learning curves and more confusion.

The second issue is much easier to mitigate than the first one. To solve the second issue, the GrowThings library adopts intuitive class names such as "TemperatureSensor" to improve beginner-friendliness and ease of instruction to free teachers from a superfluous conversation on what "DHT11" is. This solution is part of the "high-level" design of the GrowThings library aimed to improve the semantic transparency of the MicroPython code that the students write. "High-level" means packaging a series of instructions issued to the computer into one command more understandable to humans. For example, "turn on the LED" is a high-level, more human-understandable command, which consists of two lower-level instructions: 1) set the mode of the pin to which the LED is connected to "output" and 2) set the output level of the same pin to high. Using this design, the GrowThings library hides the complexities of interacting with electronic devices within a set of high-level commands that are as meaningful as the labels of the blocks in BPLs. A few more examples include the "LightSensor.get_lux()" command, which instructs the light sensor to measure the light intensity and return it as a decimal

value. The “TemperatureSensor.get_temperature()” and the “Relay.on()” commands do precisely what their names suggest. The meaning of the code thus becomes more transparent to novice learners about the actions they want the computers to perform. This design helps the GrowThings library approximate the semantic transparency of BPLs.

3. The Fail-Safety Principle

The fail-safety principle adds another layer to MicroPython to improve its beginner-friendliness. BPLs employ many mechanisms to prevent beginners from making mistakes. For example, BPLs cleverly uses colors and shapes of the blocks to indicate the compatibility between blocks visually, and incompatible blocks cannot be joined together. When mistakes do happen, BPLs also provide helpful error messages to help the users correct their mistakes. Compared with BPLs, TPLs generally incorporate fewer mechanisms to prevent mistakes and provide rather arcane error messages. While these error messages provide many technical details (such as the type of the error and the line number where the error occurred) for seasoned programmers to debug their code, the messages can be unhelpful and intimidating to untrained eyes.

Although the GrowThings library is built on top of the MicroPython programming language and thus cannot change all error messages from the base language, it does what is possible to alleviate the problem. The classes are designed to detect errors at the earliest stage. As a result, when the users create an instance and an error happens, the program will immediately report the error with user-friendly messages that include detailed descriptions of the error and recommendations for action. For example, if the microcontroller does not find a light sensor at the Port 6 (the next sections will describe in detail the Wio-Link microcontroller board and the ports it has available)

as the user specifies, the error message would read “No light sensor found at Port 6. Please check if a light sensor is connected at Port 6.” Such error messages prompt the possible actions that the users can take to correct these errors without consulting the others.

Another mistake beginners make frequently is connecting a device to a port with which it is not compatible. Because of the underlying communication protocol used for different devices, specific devices can only be used with certain ports on the microcontroller. For example, the light sensor uses a communication protocol called Inter-Integrated Circuit (I2C), which is only supported at one specific port, and the soil moisture sensor is an analog device that can only work with another port. When these devices are specified to be connected to incompatible ports, an error message will also appear and prompt the user of the ports to which the devices should be connected.

4. The High-Ceiling Principle

The high-ceiling principle entails that the learners should be able to use the GrowThings library for complex projects. While the name “GrowThings” might suggest that this library is only relevant to building smart greenhouses, the functionalities it provides, such as controlling relays and servos, reading sensor values, and displaying information on OLED screens, can be used in other projects in robotics or smart home. More importantly, the students are coding in Python, which means that they can apply the knowledge they have acquired about Python to applications as well. For example, the library for a popular tool on Raspberry Pi called SenseHAT also has an API that instructs the sensor to measure temperature. It can be used as follows:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.get_temperature()
```

Similarly, in the GrowThings library, temperatures are measured this way:

```
from sensors import TemperatureSensor
ts = TempeartureSensor(port=3)
ts.get_temperature()
```

The above example shows that the GrowThings library, compatible in philosophy with other tools built for novices to learn to program, allows those with experience in other tools to adapt to GrowThings quickly, and vice versa. Numerous popular high-level packages in the industry, such as the Scikit-Learn package (Buitinck et al., 2013) for machine learning and the Keras package (Chollet, 2017) for deep learning, can also be programmed similarly (Figure 6):

GrowThings Library	**Scikit-Learn (performing linear regression)**

`from sensors import TemperatureSensor`
`ts = TemperatureSensor(port=3)`
`ts.get_temperature()`
	`from sklearn.linear_model import LinearRegression`
	`lr = LinearRegression()`
	`lr.fit(X, y)`
	`lr.predict(new_X)`

Figure 6. A comparison between the GrowThings Library and the Scikit-Learn package.

5. The Scalability Principle

The GrowThings library is designed to be ready for other educators and researchers interested in using it to introduce computing into K-12 classrooms. To that end, the library is open-source (available at https://github.com/digicosmos86/Wio-Link/), and anyone can download and compile the code with the instruction and tools provided in the repository. The library is also built into the firmware of the Wio-Link board. The firmware, together with the instructions of installing the firmware on to the Wio-Link

board, is available at https://growthings.readthedocs.io/. The website also offers extensive online documentation of the APIs of the GrowThings library. This documentation is mainly designed for educators and more advanced students as a reference of the APIs, in case they need to lookup for detailed usages of individual devices.

The format of the documentation is designed to be consistent with those of real-world Python libraries, such as the "SenseHat" library and the documentation for MicroPython itself so that those who use the documentation will have the authentic experience of looking up API documentation. In the meantime, in order to accommodate for less technical users, a picture-based index page is also provided, so that the users can locate functionalities related to different devices. A text-based index is provided as navigation support on the left of the page to enable quick switch between pages in the documentation. In line with the structure of the library, the documentation organizes classes under different pages corresponding to the categories in which they fall. For each class, a short description is provided to briefly describe the usage of the corresponding device, followed by detailed descriptions of usages of available methods. Per convention, default arguments of the methods are available with optional arguments placed in squared brackets ("[]").

The Wio-Link Board

The "Wio-Link" development board (Figure 7) was selected as the microcontroller board on which the GrowThings library operates. Many competing computing hardware platforms exist in the educational space. The most popular of these platforms are the Raspberry Pi micro-computers, the Arduino microcontrollers, and the micro:bits microcontrollers. Appendix 2 provides an item-by-item comparison of the pros

and cons of these platforms. I considered all these platforms before choosing the lesser-known Wio-Link microcontrollers for these two reasons:

First, the Wio-Link boards have six “Grove” ports. The Grove ports are standardized physical interfaces for faster and risk-free wiring between microcontroller boards and other computing devices. External devices need to be connected to corresponding pins on the microcontroller so that the latter could “talk” to them. Without these Grove ports, students would have to wire external devices to microcontroller boards with DuPont wires and breadboards which can be an intimidating and time-consuming yet tedious and repetitive task for novices at any age level. Without a deep understanding of how electronic devices work, memorizing how to wire these devices to the microcontroller is difficult and sometimes punishing because wrong wiring can damage the microcontrollers and the external devices. The Grove ports negate the need for wiring by offering uniform ports that look the same externally. The Grove interface hides the details of the implementation of the underlying communication protocol and eliminates the risk of damaging the devices since there is only the correct way to connect devices.

Second, the Wio-Link board offers wireless connectivity within an affordable package. The board itself costs $14 approximately half of the cost of the Arduino Uno microcontroller. Despite the low cost of this board, it can be connected wirelessly to the Internet which makes it possible to move the data out of these microcontroller boards and to design innovative learning activities based on the data from these devices.

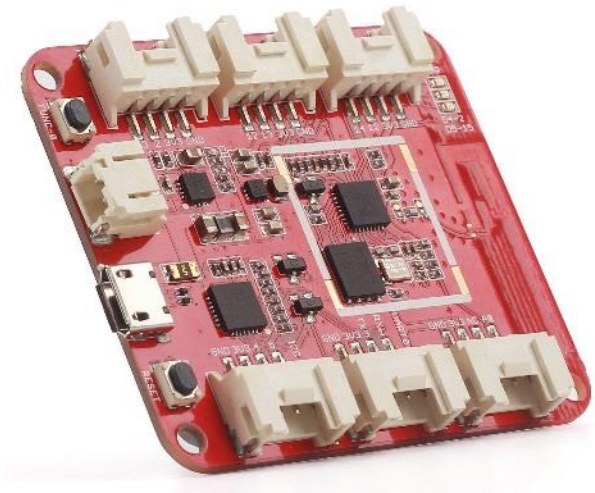

Figure 7. The Wio Link microcontroller.

Curriculum Design: Focusing on CT-in-STEM Inquiry

The curriculum used in this study is project-based (Krajcik & Blumenfeld, 2006), focusing on the science of greenhouses. The goal of the curriculum is to support upper middle school level students to use the computing tools described in the previous sections to design, build, and code automated table-top smart greenhouse. This process uses the greenhouse as a medium to engage students in scientific practices and foster computational thinking simultaneously. The curriculum also features a significant computing component that develops fundamental programming and computer science concepts in Python. It is designed for in-school settings for ten instructional days with approximately one hour per day. While the curriculum presumes an understanding of middle school science, it does not require any prior knowledge in programming from either the teachers or the students who use them. The opinions and feedback of science teachers weighed heavily in the development of the curriculum especially in the aspect of classroom instruction.

The lesson plans are accessible to both teachers and students electronically at http://growthings.netlify.com/unit1/lesson1. They are designed to double as both lesson plans for the teachers and coding tutorials for the students. The main advantage of this online format is that the programming code is well-separated with syntax highlighting from the rest of the lesson plans. This format also provides links to external resources and concepts for those who are interested. Together with the API documentation, the lesson plans and API documentation provide all the information the students need to work with the GrowThings toolkit.

The ten-day curriculum includes six instructional days, three hands-on free coding/building days, and one final day when students present and reflect on their work. (Please see Appendix C for more detail on the structure of the curriculum). The lesson plans follow backward design principles. Each lesson starts with purposes of the lesson, driving research questions for inquiry, science and computer science concepts/vocabulary, and target skills. Per the suggestions of science teachers, each computer science concept is followed by a succinct description outlining what the students need to understand. A "lesson highlights" section was also included that contains the relevant items in the state science curriculum framework, alignment with CSTA computer science standards, and the computational thinking component in the lesson. Before each day's class, students will read a two page brief focusing on the science content of the day. Instruction usually begins with a short "Do Now" section that briefly reviews the content of the previous lesson. The first 20 minutes of the lesson focuses on the science content of the day and includes whole-class or group discussions, videos, and reviewing the reading materials. The lesson then switches to an unplugged-style

instruction on how to program one or two new computing devices relevant to the science content of the day. Afterward, the students work in pairs to program with the GrowThings library and answer the research questions of the day with a few hints. They reflect on their learning experience with exit tickets at the end of the class.

The "symbiotic" relationship (Jona et al., 2014; Grover & Pea, 2017) between science and computational thinking is reflected through the way the curriculum embeds computational thinking into inquiry activities. Each lesson provides an authentic, problem-driven context for the integration of computing tools. For example, after the students learn to use the temperature sensor to measure the air temperature, the teacher will ask them, "What if I want to measure the temperature every five seconds automatically?" This question provides an authentic context for loops to be introduced and prompts students to think of using automation to solve problems.

The curriculum also introduces computing concepts incrementally which establishes the flow of scaffolding. The first lesson is an overview of both real-world greenhouses and the smart greenhouses that the students are building. Python is first introduced as a "sophisticated calculator," the same way it is introduced in the official Python manual. This first lesson provides context for students to be introduced to import statements that are essential for interacting with sensors and other devices. Lessons 2 and 3 focus on the colors of artificial grow lights and the exploration of the relationship between light intensity and distance from the light source. The colorful LED strips were designed as a hook to pique the interest of the students because they are colorful, attractive, programmable to display different patterns, and fun to play with. Using the light intensity sensor, the students will manually measure light intensity at different

distances from the light source, graph the readings, and use the graph to postulate on whether nature of the relationship.

The students will then switch in the next three lessons on the temperature sensor and use servos and relay-controlled fans to manipulate the temperature and humidity inside the greenhouse. Of the three lessons, the highlight is a "treasure hunt" activity that demonstrates the potential of integrating computational thinking for inquiry activities. Each of the three chosen students is given a Wio-Link board with a set of temperature/humidity sensor and a light intensity sensor. The sensors are uploading data to a live dashboard on the cloud, which dynamically creates line charts with the readings from the sensors in real-time. The chosen students' tasks are to hide these three boards in different areas of the class, but the rule is that the rest of the class should be able to locate these boards with what they see on the live dashboards. Before doing so, they must hypothesize what patterns they are going to see on the live dashboards. The rest of the class then also hypothesizes where the boards are located based on the graphs on the live dashboards before trying to find them. The activity models the inquiry process for the students while the computing tools automatically handle data collection and visualization. The live dashboard in real-time makes it easier for the students to make sense of the data as well as creating opportunities for innovative and exciting learning activities.

Alignment of Design with Frameworks

The design process results in a set of computing tools that aligns well with the design frameworks mentioned in the literature review. The design follows all six principles in Repenning et al.'s (2010) framework:

1. **Low threshold:** the high-level GrowThings library based on the beginner-friendly MicroPython TPL and the Grove ports on the Wio-Link board make it easy for beginners to interact with microcontrollers and external devices.
2. **High ceiling:** learners can use the computing tools to build complex projects, and their knowledge gained in this process can be applied to industrial-strength applications with Python.
3. **Scaffold flow:** the GrowThings library and curriculum introduces new variables and concepts incrementally, enabling concepts in latter classes to build on those learned previously.
4. **Enable transfer:** greenhouse science applies to other contexts as well, and so do programming skills in Python.
5. **Support equity:** the computing tools are open-source and affordable. The integrated approach of computing and science reaches a broader audience.
6. **Systemic and sustainable:** the GrowThings toolkit can also be used to teach other concepts in science and other STEM disciplines such as engineering design and mathematics.

The greenhouses are also micro-worlds modeled on real-world greenhouses that provide authentic simulations that do not strip away the complexity of real-world inquiry, and the computing devices provide visually enhanced data analysis and modeling (Windchitl, 2000). There are also ample opportunities for scaffolding (Table 3):

Table 3

Implementation of the Inquiry Scaffolding Framework (Quintana et al. 2014).

Scaffolding Guidelines	Scaffolding Strategies	GrowThings Implementation
Science inquiry component: Sensemaking		

Guideline 1: Use representations and language that bridge learners' understanding	1a: Provide visual conceptual organizers to give access to functionality	A visual "cheat sheet" was provided for students to locate functionalities of the GrowThings library
	1b: Use descriptions of complex concepts that build on learners intuitive ideas	Describing class organization as "folders" and variable names as "nicknames"
	1c: Embed expert guidance to help learners use and apply science content	N/A
Guideline 2: Organize tools and artifacts around the semantics of the discipline	2a: Make disciplinary strategies explicit in learners' interactions with the tool	Explicitly explain what scientists do
	2b: Make disciplinary strategies explicit in the artifacts learners create	Learners create computational artifacts for the collection of scientific data
Guideline 3: Use representations that learners can inspect in different ways to reveal important properties of underlying data	3a: Provide representations that can be inspected to reveal underlying properties of data	Provide automatic data visualizations
	3b: Enable learners to inspect multiple views of the same object or data	The juxtaposition of multiple live data visualizations
	3c: Give learners' malleable representations that allow them to directly manipulate the representations	Provide a website for learners to look up R, G, B codes to design different colors of their LED strips
Science inquiry component: Process management		
Guideline 4: Provide structure for complex tasks and functionality	4a: Restrict a complex task by setting useful boundaries for learners	Limiting the number of sensors and variables measured
	4b: Describe complex tasks by using ordered and unordered task decompositions	Using guiding questions for students to complete programming tasks
	4c: Constrain the space of activities by using functional modes	N/A

Guideline 5: Embed expert guidance about scientific practices	5a: Embed expert guidance to clarify the characteristics of scientific practices	Modeling the process of hypothesis, data collection, and answering questions
	5b: Embed expert guidance to indicate the rationales for scientific practices	N/A
Guideline 6: Automatically handle nonsalient, routine tasks	6a: Automate nonsalient portions of tasks to reduce cognitive demands	Automating data collection/visualization, high-level library design simplifies the process of interacting with microcontrollers
	6b: Facilitate the organization of work products	A visual "cheat sheet" was provided for students to locate functionalities of the GrowThings library
	6c: Facilitate navigation among tools and activities	Online documentation provided and the lesson plans provided as tutorials
Science inquiry component: Articulation and reflection		
Guideline 7: Facilitate ongoing articulation and reflection during the investigation	7a: Provide reminders and guidance to facilitate productive planning	Provide worksheet for the design of greenhouses
	7b: Provide reminders and guidance to facilitate productive monitoring	N/A
	7c: Provide reminders and guidance to facilitate articulation during sense-making	Do now activities and exit-ticket questions use verbal and written articulation to reflect on what is learned
	7d: Highlight epistemic features of scientific practices and products	N/A

Summary

In this chapter, I described the design of the computing tools that fit under multiple design frameworks for both computational thinking and scientific practices. In

the next chapter, I will describe how this conjecture map will guide the data analysis of the study.

Chapter 4: Methods

This chapter describes the multiple-case study design that focuses on investigating how the design of the computing tools and the curriculum supported embedding computational thinking into middle school classrooms' science instruction. The purpose of this study is to critically examine how the ideas that undergird the design translated into classroom practice and to provide guidelines for integrating computational thinking into science classrooms. The research questions are:

1. How did the teachers implement and reflect on their instruction in this learning environment?
    a. How did they understand the interplay between computing and science?
    b. What instructional practices did the teachers utilize?
    c. What were their challenges adapting the design of the tools into their own instruction?
2. How did the students engage with computing and science in the learning environment?
    a. How did the students make connections between coding and science in this environment?
    b. What were their challenges engaging in learning to code and learning science through coding?

Research Design and Rationale

This study follows the design of an exploratory, qualitative, multiple-case study to address the two research questions above. According to Yin (2009), case studies are preferable for studies that 1) pose "how" or "why" research questions that are exploratory

or explanatory; 2) grant researchers little control over events; and 3) focus on a contemporary phenomenon that is difficult to separate from its real-life context. This study poses two questions that explore "how" the design of the computing tools influenced the experiences of a specific group of students and their teachers learning science and developing computational thinking skills, which is a contemporary phenomenon happening in a real-life context. The experiences of the participants were not only intricately connected to the design of the tools but also affected by their prior exposure to and interest in computing and science, which is also shaped by the school and community they were in. It is impossible to separate the phenomenon from the context. Within the intervention itself, the researchers were unable to and did not manipulate the events that happened in this context or determine the subjects of intervention or the amount of intervention received. Therefore, a qualitative multiple case study that evaluates the design of the computing tools is most appropriate for an in-context and in-depth approach.

Case studies are commonly criticized for potential issues that could threaten external validity: how can the findings of a case study generalize beyond the case at hand (Yin, 2009). The inclusion of multiple cases specifically addresses this concern. In the next sections, I describe the two cases that I study, each focusing on one science teacher and his/her students. The two cases contrast each other because of the different characteristics of the teachers, which to some extent, alleviates the arguments against the single-case design over the "uniqueness or artificial conditions surrounding the case" (Yin, 2009, p.61).

Methodology

Research Context

The study uses data from a larger interventional study in an urban-setting public middle school in Northeastern US. Public enrollment data show that the school serves over 600 students, with the majority being Hispanic (45.6%) and White (39.6%). Black and Asian populations are much smaller (8.9% and 3.6%). A total of 33.6% of the students were classified as economically disadvantaged, 10.5% were English Language Learners (ELLs), and 19.5% received special education services. The city, on the other hand, has sizable populations of Hispanics (12.8%) and Asians (10.4%), both well above state averages.

All of the school's 198 eighth graders and their two science teachers participated in the larger intervention. They were the study's pool of subjects from which cases were selected. Student assent, parent consent, and teacher consent were requested from all individuals participating in the intervention, but not all agreed to participate or gave full consent to be video or audio recorded. The unconsented students remained in the same classrooms and received the same instruction, but no research data were collected from them. Table 4 describes the demographic information of the 198 eighth-graders from whom the cases were selected. Of the 193 students who have completed a demographic survey, 95 (49.2%) identified as male and 94 (48.7%) as female, while 4 (2.0%) identified as non-binary. 70 (36.3%) are White, 88 (45.6%) are Hispanic/Latino, 35 (18.1%) identified as African Americans, Asians/Pacific Islanders, Native Americans, and Two or More Races.

Table 4

Demographics of Students by Teacher, Gender, and Ethnicity (N=193[a])

			Ms. Petralia	Mr. Hanrahan	Total
White	Gender	Male	21	19	40
		Female	17	13	30
	Total		38	32	70
Hispanic/Latino	Gender	Male	23	21	44
		Female	20	22	42
		Other	1	1	2
	Total		44	44	88
Other	Gender	Male	6	5	11
		Female	9	13	22
		Other	0	2	2
	Total		15	20	35
Total	Gender	Male	50	45	95
		Female	46	48	94
		Other	1	3	4
	Total		97	96	193

[a] five students' demographic information is missing.

The classrooms of the science teachers were located in two separate wings of the school building. Each teacher had four instructional blocks – A, B, C, and E, each with roughly 25 students. The school had a separate instructional calendar of six-day cycles and each teacher taught the same content to all four blocks in a particular order, according to which instructional "day" it was, usually with two in the morning and two in the afternoon. Planning periods for the science teachers happened between morning classes as well as between afternoon classes.

All research activities in this study occurred during these regular in-school instructional "blocks" within the three weeks between the end of state standardized tests and the end of the school year, a time when both teachers and students were relieved from the pressure of standardized tests. Due to the end-of-year activities interspersing the

three weeks that occurred during instruction time, the actual number of days on which instruction might occur was approximately 13, which includes two or three catch-up sessions at the teachers' discretion where no new content was introduced. This allowed some flexibility for the 10-day curriculum to be fully implemented.

Participants

Two cases were constructed from the two science teachers and the 198 eighth-graders who participated in the larger study. Each case consists of one science teacher and eight students purposefully selected from different blocks following a procedure detailed in the next section. The two science teachers had different characteristics. Ms. Petralia, was a white female in her late 20s with a chemistry background, while the other, Mr. Hanrahan, was a white-coded male in his early 50s with a background in biology. While they differ in years of experience teaching science, neither indicated any non-trivial experience with programming. Ms. Petralia did mention her experience using HTML, a markup language generally not considered a programming language, to customize her MySpace homepage, but she also noted that it happened years ago, and the experience was drastically different from using Python to control electronic devices.

During the research study, the students worked in pairs of two. They were given choices of the partners that they want to work with, which were then taken into consideration when teachers made their final pairing decisions. However, the teachers also considered the prior experience and potential dynamics between students as important criteria. The resulting groups consisted of pairs of students who had similar prior knowledge and worked well together. While most of the students had little prior experience programming in Python, those whose teachers consider to be much better

versed in coding than the rest were allowed to work on their own if they so petitioned to. Some students were allowed to work in trios at the teachers' discretion.

Sample and Sampling Design

The overall pool of potential subjects consisting of the two science teachers and 198 eighth-graders in the larger study was achieved through availability and convenience. The teachers had worked with the research team in the past, and the administrators of the school and district valued and supported the teams' research. This group of individuals was fairly representative of the population of the city in which the school was located and of public schools in working- to middle-class suburban settings. The teachers of this study were also representative of the science teachers in having little prior experience with working with computing. The two teachers' differences in genders, personalities, educational backgrounds, teaching experiences, and styles make them potentially interesting subjects for comparison.

The selection of students into each case led by a teacher followed a stratified purposeful sampling procedure (Figure 8). Before the research, the research team administered a demographic survey to obtain information about students' self-reported gender, race, comfort with coding, interest in coding, and skill levels with coding. Using these criteria, the research team selected eight pairs of students (16 in total), four pairs from each teacher, for observation of their group work on designing greenhouses and for artifact-based interviews (Brennan & Resnick, 2012, see data sources section for interview protocols). The selection focused on maximizing the variability in each of the selection criteria to achieve a group of individuals with broad distributions of comfort, skills, and interest in coding. The research team also deliberately over-sampled from

under-represented gender and racial/ethnic groups to study more about the students from these groups.

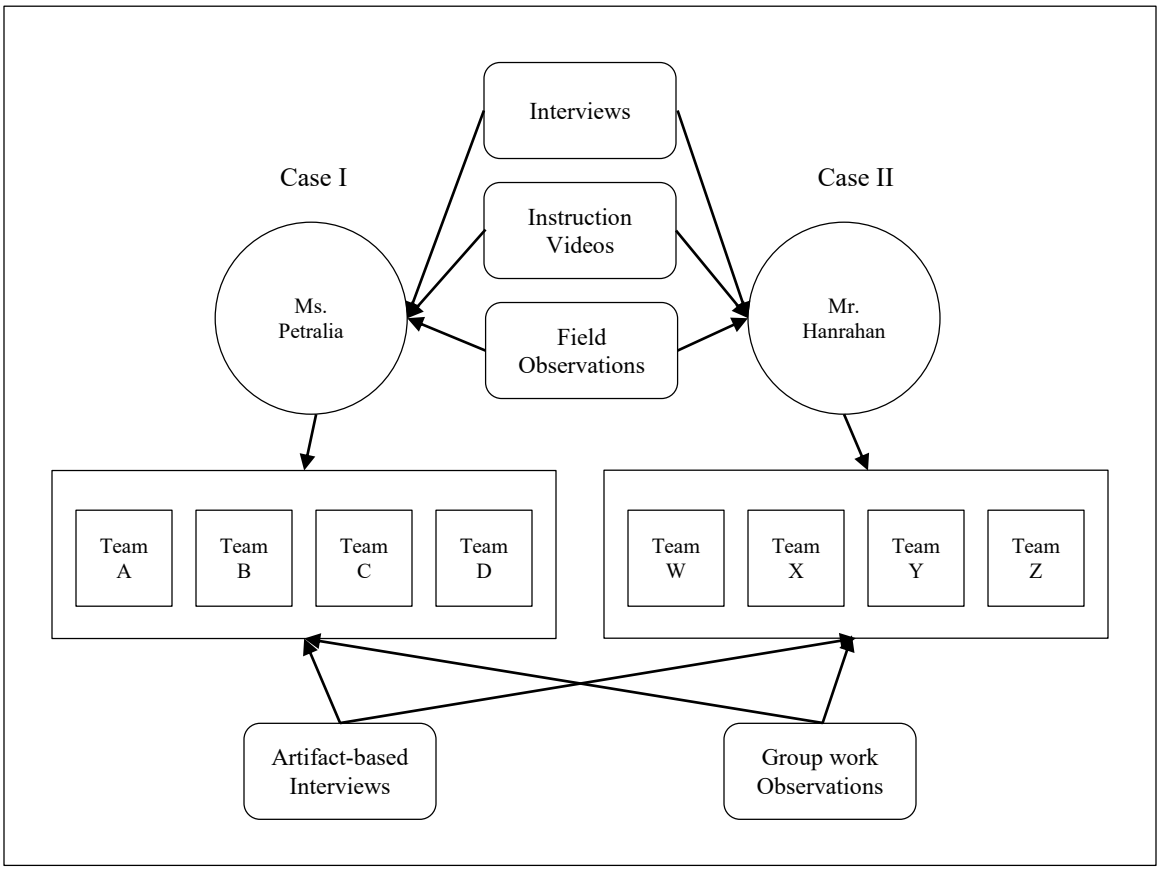

Figure 8. Case construction and data collected from each case.

Data Collection

The intervention happened in three phases (Table 5): a two-week PD Phase, an approximately two-week Instruction Phase, and a one-week Design Phase. Data collection occurred before, during, and after these phases of the research study.

The **PD Phase** happened within the two weeks before the first day of instruction during the two teachers' planning time. It focused on 1) familiarizing the teachers with the Wio-Link microcontroller, other computing devices, and the GrowThings library, and 2) working with the teachers to design the curriculum and instructional activities. For

approximately twice a week, the research team met with the teachers, during which time they introduced the teachers to the technologies, seeking their feedback on the design of the computing tools. In the meantime, the research team also discussed with the teachers about the structure of the curriculum and the instructional activities. With the teachers' input, the research team worked on adding support materials that teachers deemed helpful for student learning. Teachers' feedback was immediately incorporated in the online lesson plans, and the updated versions presented to the teachers during the subsequent meeting for more feedback. To reduce the teachers' anxiety and ease them into using and teaching technologies with which they were not comfortable, the PD Phase was intentionally designed to be informal and unstructured with minimal elements of research involved. The researchers only took retrospective notes after each session.

Table 5

Timeline for the research study.

Time	Phase	Instructional / Learning Activities	Data Collected
2 Weeks 2-3 Hours/Week	**PD Phase**	The team the teachers on how to use the GrowThings library to program the Wio-Link microcontroller and the sensors during their planning time. While the teachers were familiarizing themselves with the programming, the research team also worked with them to improve the design of the curriculum, specific instructional activities, and the supporting materials for instruction.	• Retrospective field notes
Last day of state standardized testing			
9 Days 6 Days of New Content, 1 Hour/Block/Day	**Instruction Phase**	On each instructional day, the teachers taught one lesson of the curriculum. (Please see Appendix C or http://growthings.netlify.com/unit1/lesson1/) for detailed lesson plans. Teachers also used non-instruction days to catch up or review content learned in previous instruction days.	• Teacher Interviews (after Day 1) • Instruction videos • Observation notes

3 Days 3 Instructional Days, 1 Hour/Block/Day	**Design Phase**	The students designed, built, and coded their smart greenhouse. The teachers set expectations and rubrics for greenhouses.	• Student observation videos • Observation notes
1 Day	**Gallery Walk**	The students showcased their greenhouses and presented their finished work to adults and peers, explaining how they worked and describing their experience with the project.	• Fieldnotes
Post-Interview (Artifact Based) with Students Post-Interview with both teachers			

The **Instruction Phase** happened within the first two weeks of the 13-day window between the state standardized tests and the last day of school. Interviews of the teachers happened after Day 1 of this phase. Due to holidays, school plays, and other activities, the students received approximately 6 instructional hours of new content. While the teachers each planned different lessons, the content and structure of instruction approximately followed the sequence of the first six lessons of the online lesson plans described in Chapter 3 (please refer to Appendix C or online at http://growthings.netlify.com/unit1/lesson1/): on each day with new content, students were introduced to the science behind the greenhouses and then the relevant sensors, actuators, or display devices that they could use to control the corresponding variable of the greenhouse. The students had 10-20 minutes to free-program those devices and complete additional coding challenges each day. At any point during the Instruction Phase, at least one member of the research team was in each teacher's classroom for technical and instructional support while at least one other research team member filmed the instruction and took notes during two of the four instructional blocks every day selected by the teachers.

The **Design Phase** followed immediately after the Instruction Phase. In the Design Phase, the students built their smart following within parameters set by their teachers. The teachers designed rubrics for the final greenhouses and provided guiding questions. Then the students designed their greenhouses in their workbooks and explained how the data that they would potentially collect from the greenhouses could address their research questions. The research team observed the students from the eight focus groups and audio- and/or video-recorded their discussions, occasionally asking probing questions.

On the last day of the research study, the students demonstrated their finished greenhouses in a gallery-walk activity where students from other grades and adults could ask them questions about their greenhouses. Post-interviews of the teachers and students happened after the last day of the design phase.

Data Sources

According to Yin (2009), case study research typically faces the threat of construct validity – "a case study investigator fails to develop a sufficient operational set of measures and that 'subjective' judgments are used to collect the data" (p.41). He recommended the use of multiple data sources to address this concern. Since each case is defined as a science teacher and the student he/she teaches, multiple types of research data will be collected from both the teacher and the students for each case in order to achieve an in-depth understanding of students and their teachers' learning experience the development of their computational thinking skills. The following data will be collected from the teachers, and all non-textual data will be transcribed into texts and imported into NVivo 12 for data management and analysis.

Teacher semi-structured interviews. These interviews occurred at two time points for approximately 45 minutes each, all of which will be audio-recorded and transcribed. The first interviews happened after Day 1 of the Instruction Phase. The research team asked about the teachers' prior experience, problems anticipated during instruction, and their opinions about embedding computing in science classrooms. The post-interviews happened right after the Design Phase. The teachers reflected on their experience teaching computing and science and gave feedback on what they consider would be helpful in the next round of research. The protocols for these interviews are in Appendix A.

Teacher instruction videos. On each day of the Instruction Phase, the research team videotaped each teacher's instruction in one block chosen by the teachers as representative of their teaching. The team also filmed the same block during the Instruction Phase. Each video lasts approximately the length of one class session (55 minutes). Due to technical issues, the instructions from both teachers on Day 1 are missing. The research team was not onsite for the remedial lessons, so these lessons were not recorded, either.

Teacher observation field notes. One member of the team also took observation notes, documenting the instructional goals of the class, instructional activities of the class, notable things that happened in the classroom, interesting comments, as well as the note-takers' comments on the day's instruction. These notes were used to triangulate with the video records. Observation protocols are also included in Appendix A.

To understand the learning experience from the students' perspective, the research team also collected data from the 16 students in the sample. Because of the sampling

procedure to include students with varied backgrounds and interests, they were not all from the same block in which the teachers' instructions were filmed. These data include:

Student artifact-based interviews. These interviews are approximately 10-15 minutes each and happened after the project concluded. The students had their greenhouses in front of them and answered questions about how they designed their greenhouses, how their experiences were similar to or different from their expectations, and whether their opinions and perceptions about science and computing have changed after their experience in the intervention. These interviews were audiotaped and transcribed. Protocols for the interviews are included in Appendix A.

Student observation videos. During the 3-day Design Phase, at least one member of the research team sat with each selected pair/trio of students, observed their collaboration and sometimes provided assistance or asked probe questions. The collaboration was videotaped and transcribed, and if possible, the students' activities on-screen were captured with Camtasia. Each video will be approximately the length of one session (55 minutes). The observation protocol is included in Appendix A.

Data Analysis

The goal of the data analysis is to distill and interpret evidence from multiple types of qualitative data to address the two research questions. The conceptual framework (Figure 3) constructed in Chapter 2 will guide the analysis and provide theoretical foundations for coding schemes and, if possible, for making claims of causality and generalizability. The analysis will follow Creswell's (2013) recommendation of four steps to construct an in-depth portrait of a case study: 1) a detailed description of each case and its context, 2) within-case theme analysis, 3) cross-case theme analysis, 4)

discussion and generalizations. Following these steps, I will construct detailed profiles of teacher instruction and student learning as two separate cases and compare and contrast these cases in the data analysis. The rest of this chapter describes the organization of the results of the data analysis in the next chapters and the analytical procedures taken to investigate the qualitative data collected.

Description of Cases and Context

Yin (2009) and Stake (1995) both contended that case studies focus on contemporary phenomena that cannot be separated from the naturalistic environment in which they occur. Therefore, for each case of this study, I will provide detailed descriptions of its context and the subjects. I will present a portrait of each teacher, illustrating their personalities, academic backgrounds, experience and philosophy in teaching, and prior experience with technology in more detail than given in this chapter, using evidence from the data. I will also describe the students in each case, describing their personalities, group dynamics, prior experience in coding, interests, and beliefs in coding and science.

Within-case Theme Analysis

In this step, I will build for each case an in-depth narrative of what happened in each teachers' classroom. With a visualization of the main themes, which illustrates how the themes in each case fit together to answer the sub questions of RQ1, I will start in each case by describing how each day of instruction looked like for the teacher. I will then describe the teacher's understanding of computing and science in the learning environment (RQ1a) and his/her instructional practices, including the instructional languages used, the learning activities students participated in, and the interactions with

the students (RQ1b). The challenges the teacher encountered in each case will be presented next (RQ1c), which tend to be jointly influenced by the teacher's background, his/her perception of the learning environment, his/her instructional practices, and the design of the tools and curriculum.

In order to answer RQ2, which focuses on the learning experiences of the students in the learning environment, I will use a similar structure. Guided with a visualization of the themes, I will describe the students in each case, followed by themes in their understandings of computing and science, which respond to RQ2a. The challenges that the students faced differed slightly in each case, which will also be presented to answer RQ2b.

Cross-case Theme Analysis

According to Miles and Huberman (2014), the purposes of cross-case analysis are to enhance generalizability and transferability and to deepen understanding. In the cross-case analysis, I will juxtapose the similarities and differences across the two cases. The focus will be on weaving together the themes across the cases and building a narrative on how the two teachers' different backgrounds with computing, instructional focuses, and conceptualizations of the computing-science relationship might have contributed to the slightly different outcomes in each case.

Discussion and Generalizations

After single and cross-case analyses, I will focus on using the themes and narratives constructed in the previous chapters to answer the following questions: 1) how can the teachers' perceptions of the relationship between computing and science in this learning environment be used to enrich the theoretical underpinnings of the integrated

approach to computing and science? 2) How do the lessons from the implementation of the Smart Greenhouse project inform future work with the science teachers? 3) How can the design of the GrowThings library be improved after examining how the five principles worked in practice? With these questions answered, I will recommend future directions for research and practice.

The Two-cycle Coding Process

I followed the two-cycle coding process that Saldaña (2013) and Miles and Huberman (2014) recommended and divided qualitative coding into two stages: First Cycle and Second Cycle coding. First Cycle coding focused on the initial assignment of meaning to chunks of data, and Second Cycle coding focused on extracting and generalizing patterns and themes to address the research questions. With this Two-Cycle process of coding, the data analysis examined data iteratively in each cycle in order for new understanding and interpretation of the data to develop and for more information from the data to be synthesized.

First Cycle coding. Both content coding and affective coding were the two main coding methods used in this process. Before starting, I had a set of pre-existing codes set up for each research question and its sub questions, but the process featured more open-ended discovery and resulted in a coding scheme much richer than the original set of codes. Table 6 summarizes the coding scheme and provides sample codes used in this study.

Since RQs 1 and 2 had separate focuses on the teachers and students, I had separate categories for teacher codes and student codes. Within teacher codes, I had "understandings" and "connections" categories for RQ1a, which focus on how the

teachers perceived the learning environment and how they made connections between computing and science. The former included codes not only on how the teachers described computing (coding) and science separately but also on how they described their instructional goals and focuses as part of their perceptions of the learning environment. The latter consists of codes of the direct connections teachers made between computing and science and beyond, such as connections between computing and ELA, social sciences, and real life. The main evidence used for this sub-question was the teachers' semi-structured interviews.

Table 6

The coding scheme and sample codes.

RQ1. How did the teachers implement and reflect on their instruction in this learning environment?		
Sub-Question	**Sample Codes**	**Data Sources**

1a. How did the teachers understand the interplay between computing and science?	• Understanding
	  ○ Computing
	  ○ Science
	  ○ Goal/Focus
	• Connection
	  ○ Computing-Science
	  ○ Computing-ELA
	  ○ Computing-Social Sciences
	  ○ Real-life
		• Teacher interviews
		• Fieldnotes

1b. What instructional practices did the teachers utilize?	• Language
	  ○ Syntax
	  ○ Semantics
	  ○ Spelling
	  ○ Repetitions
	  ○ Novice-friendly
	• Mistakes
	  ○ Typos
	  ○ Misconceptions
	  ○ Consulting research team
	  ○ Use mistakes as teaching opportunities
		• Teacher instruction videos

1c. What were their challenges adapting the design of the tools into their own instruction?	• Challenges ○ The Python language ○ The MCU ○ Lack of support ○ Timing ○ Logistics ○ EsPy IDE	• Teacher interviews • Teacher Instruction videos
RQ2. How did the students engage with the coding and science in the learning environment?		
2a. How did the students make connections between coding and science in this environment?	• Computing ○ Import ○ Loop ○ Conditional • Science ○ Plant growth-airflow ○ Plant growth-humidity ○ Plant growth-temperature • First-response ○ Coding ○ Science ○ Engineering design • Connection ○ Coding-science ○ Real-world ○ Career	• Artifact-based interviews
2b. What were their challenges engaging in learning to code and learning science through coding?	• Challenges ○ The Python language ○ The MCU ○ Lack of support ○ Timing ○ Logistics ○ EsPy IDE	• Artifact-based interviews • Observation videos

The teachers' instructional videos supported the answers to RQ1b, for which I created a separate "instruction" category. Within this category I used a broad umbrella term "language," under which I filed codes describing the content of the teachers' instruction, such as "syntax," "semantics," and "spelling", as well as codes delineating the styles of the teachers' instructional languages, such as "repetition" and "novice-friendly." Another major category was "mistakes," which focused on all types of mistakes that the teachers made as novices, such as "typos" and "misconceptions," and

the way they treated their mistakes, such as “consulting research team” and “using mistakes as teaching opportunities.” The teachers also made connections either between computing and science or between computing and the real life during their instruction. Codes for these connections were temporarily filed under “connections” in “teacher instruction,” but was later merged to “connections” category described in the previous paragraph.

Answers to RQ1c on the challenges the teachers faced came from both teacher interviews and instructional videos. Although I anticipated that the teachers would describe their challenges with the computing tools and the curriculum, the analysis revealed much more, and all codes were filed under a “challenges” category.

Besides these categories with codes that would directly answer RQ1, I also coded the teachers’ direct remarks on the design of the computing tools and curriculum under the “design” category and the adjectives that they used to describe their emotions under an “emotions” category. The emotional codes in the latter, such as “stressed,” “pleasantly surprised,” and “challenged” helped painting a holistic and vivid picture of their experiences teaching in the learning environment and were relevant to answering all sub-questions in RQ1.

Although RQ2 are similarly structured to RQ1, the coding scheme used to analyze data collected from students was different, especially for RQ1a. Because the research team anticipated the possible difficulties for novice learners in the Eighth Grade to directly articulate their perceptions of the interplay of computing and science in this learning environment, the artifact-based interviews probed their understandings by asking them what they have learned, what they were most proud of during this project, and how

they would describe their finished greenhouses (Appendix 1). Therefore, separate categories of "computing" and "science" were established to hold codes used to document what the students described they have learned about each subject. Separate "first response" and "most proud of" categories recorded the aspects of the project that were most salient to the students, and the codes in the "connections" category also captured the direct connections some students did make between computing and other school subjects and their daily life. For RQ2b, the "challenges" category was also used to document the challenges that the students encountered. Again, the "design" and "emotions" categories were also used for remarks directly related to the design of the computing tools and the curriculum and the emotions the students experienced in this learning environment.

Second Cycle Coding. The goal of this process was to revise the initial codes, establish patterns, and weave patterns into themes. The patterns examined specifically included: 1) Did the teachers' understandings of the computing-science relationship change at the beginning and the end of the project? If yes, how did they change? 2) Did the teachers' instructional goals, focuses, and practices differ? 3) Did the teachers encounter the same or different challenges? 4) How were the teachers' and the students' conceptualizations of computing and science connected? 5) Did the students from different teachers report different learning outcomes? How was that connected to the teachers' differences? These themes and patterns were organized into diagrams and visualizations that will be presented in the next chapters.

Researcher Positionality

In qualitative research, the background and experiences of the researcher

inevitably introduce bias to the way he/she interprets research data. I consider my double identity as an educator and a self-learned computer scientist to be potentially beneficial for this research. Without formal training, I am more open-minded to other pathways into computer science and alternative ways of thinking and problem-solving. Having undergone the experience of learning to program from scratch recently, I can better empathize with novice learners about their struggles. Therefore, I am confident about developing a beginner-friendly curriculum for my participants. However, as an Asian male and a member of the dominant group in the computer science industry, I might be more oblivious to the experience and needs of students from underrepresented populations, such as girls and students of both genders from minority groups. In the meantime, as the creator of the tools and curriculum used in this study, I might be biased towards its benefit and neglect the potential issues it might cause for learners. Therefore, I will deliberately focus more on the impact of this curriculum on students from underrepresented populations and present a more balanced view of the limitations of computing in supporting the integration of computing in science classrooms.

Summary

In this chapter, I described the methodology that will be used to analyze the qualitative data that I anticipate to collect from the research study. I detailed my rationale for adopting an exploratory, qualitative, multiple case study format and illustrated the process that I will use to collect, process, and analyze data. I also stated my positionality as a researcher and the lens through which I will interpret the data. The next chapters will present the results of the data analysis.

Chapter 5: The Case of Ms. Petralia

In this chapter, I present the case of Ms. Petralia and her students. I report the themes for Ms. Petralia and her students and answer the two research questions separately. RQ1, with its three sub-questions, focuses on Ms. Petralia's conceptualization of computing and science in this learning environment, her instructional practices, and the challenges she encountered while implementing the integrated approach. The first half of this chapter is devoted to reporting themes in these respects. RQ2 and its two sub-questions focus on Ms. Petralia's students' understandings of the computing-science relationship in this learning environment and the challenges they encountered. Themes related to these questions are reported in the second half of this chapter.

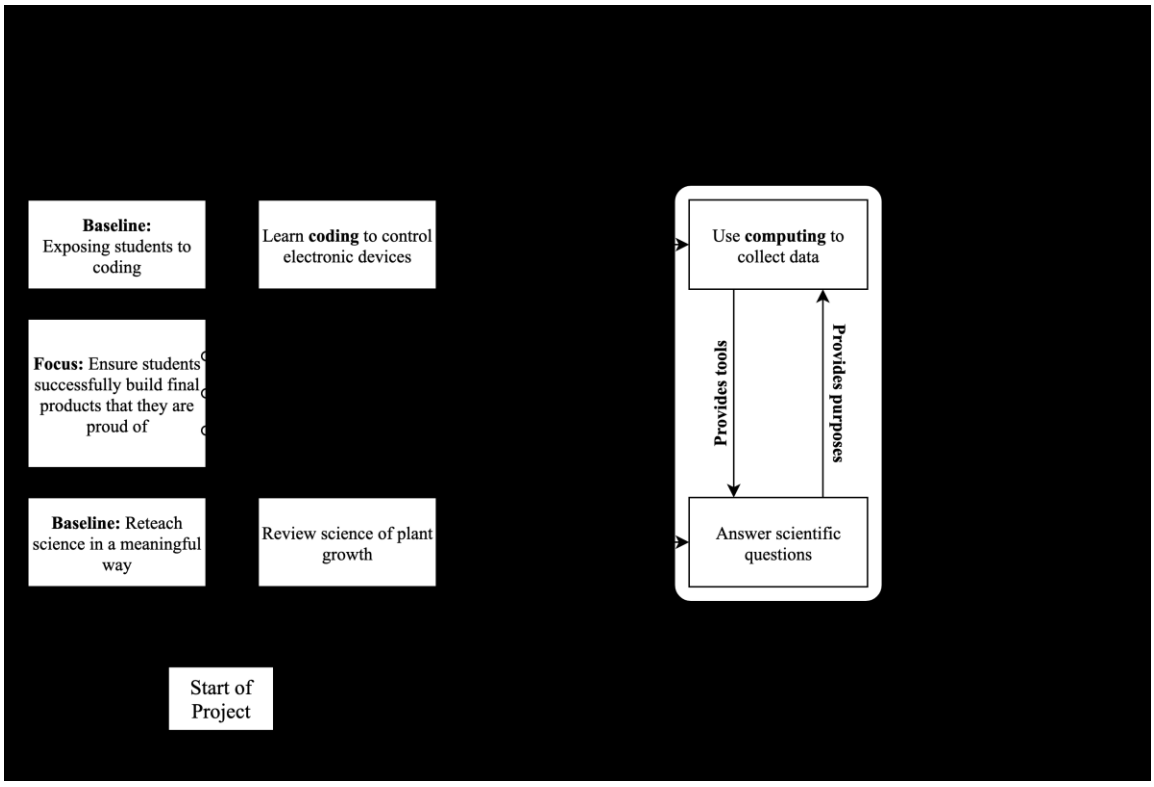

Figure 9. The thematic map for Ms. Petralia's case.

Ms. Petralia

The purpose of this section is to answer RQ1: How did Ms. Petralia implement and reflect on her instruction in this learning environment? Figure 9 maps the themes of this chapter and the relationships between these themes. The report of Ms. Petralia's case begins with detailed descriptions of her background and initial experience with this project. The subsequent sections cover her goals, instruction, and perceived outcome of the project. The rest of this section features themes that directly answer the three sub-questions of RQ1.

Background

Ms. Petralia, a young white female in her late 20s, taught science to approximately 100 students, half of the Eighth Grade of Central Middle School. Her classroom was on the east end of the hallway on the third floor. Columns titled "Vocabulary" and "Students will be able to" marked her whiteboard on the left and "Agenda" on the right, while the screen of the projector, which she utilized heavily during her instruction, covered the center portion of the whiteboard. Each day, four groups of students, each consisting of about 25, cycled through her classroom according to a six-day instructional routine. The students worked in designated pairs and sometimes in trios on a laptop at each table with an electronics kit consisting of the Wio-Link microcontroller board and all other smart greenhouse devices. Before each class, Ms. Petralia would distribute the laptops and the box of electronics to each table by herself.

This year marked the third in which she worked with Mr. Hanrahan to teach Eighth Grade science, although she had also previously taught at the elementary level. With academic training in chemistry, she self-reported slight unfamiliarity with some

specificities of the reading materials on plant growth even though she was more than equipped to cover every science standard at the eighth-grade level. However, a quick read through the articles immediately brought her up-to-date, which she described as an exciting learning experience.

According to her account, she also had had no coding experience before the project other than using HTML to decorate her MySpace homepage, which does not amount to any substantive prior experience with programming in Python. She had limited knowledge about computational thinking both before and after the project. She thought, "[computational thinking was] just being able to think in coding basically. People think in the mindset of the computer or Python if we're specifically talking about that program." Her idea epitomized how ambiguous the concept of computational thinking was to practitioners who were not specially trained to understand what it was.

Despite not having previous experience with Python, Ms. Petralia was the quicker of the two teachers to grasp the fundamentals of Python in preparation for the upcoming Smart Greenhouse project. Within the first few hours of the PD Phase, she was able to not only digest what was presented to her but also help Mr. Hanrahan and other adults in the room, explaining in her own words what each line of the code meant. She reported being "excited" about her ability to learn quickly and taking home a set of the greenhouse kit to demonstrate to her family what she learned. Enthusiastic about the project, she told her students in advance about the Smart Greenhouse project and gave her greenhouse kit to some of her students to figure out independently how to program in Python to play music with a buzzer with the help of a cheat sheet, which summarized all information students needed to start coding in Python in this project.

Ms. Petralia's Instruction

Goals. Due to the experimental nature of the project, it was difficult for the teachers to set a definitive instructional goal at the very beginning of the project. However, both had a minimal set of objectives, or baseline goals, for both computing and science that they wanted to accomplish. Although Ms. Petralia indicated to the researchers that coding and science content "has to be 50/50" in the learning environment, her baseline goal at the beginning of the Smart Greenhouse project was to expose her students to coding. She said:

> So my goal for the greenhouse unit is for students to be successful with each module, be able to turn [the electronic devices] on and off and use them successfully … the whole idea is they're interested in it, they like it, they have some desire maybe to do robotics in high school or whatnot…

This focus on coding might have resulted from her understanding of the students' backgrounds in coding and science. She perceived that although their experiences in coding differed, most students had had little exposure, especially with physical computing with a TPL like Python. Therefore, it was a "privilege" for the students to have the exposure.

For science, however, Ms. Petralia's baseline goal was "almost just re-teaching it in a way that is meaningful to them." She believed that even though the students were not entirely familiar with every aspect, they should already be knowledgeable about scientific concepts such as photosynthesis and radiation. Her everyday practice provided the best indication of how she re-taught science in a meaningful way. First, she used science questions as "hooks" to begin each class. In her opening "Do NOW!" activities every

day, she would typically include two questions, one as a refresher on the code that the students had encountered previously and the other on science from the two-page reading that students finished before the lesson. Below are two "Do NOW!" questions that she used:

1. If we have a temperature sensor (TemperatureSensorPro) connected to port 3, how would you write code to read temperature and humidity from the sensor?
2. How does the temperature affect the rate of chemical reactions?

After revealing the answer to the coding question, which the students had to write down in their composition notebooks, she would initiate a brief discussion on the science question, which would lead to a reason why the electronics and coding were necessary for greenhouses. For example, for Question 2 above, Ms. Petralia reviewed that temperature affects the rates of chemical reactions, and since photosynthesis is a chemical reaction, plants need optimal temperatures to prosper. Therefore, it would be desirable to control the temperatures inside greenhouses and learn how to read the temperatures from the temperature sensor, which segued naturally into the instruction for the day on temperature sensors and reading temperatures from the sensor:

> So [temperatures] either benefit the plants or be detrimental, hurtful to the plants. So we need an optimal range. We need an ideal situation. If it's too hot, they won't survive, and if it's too cold, they won't survive. Certain plants do better with more shade, certain plants do better with more sun. These conditions are what we need to monitor when we start putting your greenhouses together this week.

Focus. With their baseline goals, the teachers were clear on what to focus on to support the students to reach these goals. Ms. Petralia's instructional focus was on

ensuring that each student had "a finished product at the end so that they have something to be proud of to show." She said:

> I wanted them to have finished products because I feel like it means a lot to them to be proud and to show that they accomplished something. And it's a little disappointing when they plug it in and it doesn't work or something's broken or if this person has theirs all together but I don't, and it's like... Also, is that person smarter than me? We get into all these self-conscious thoughts when you don't complete something. … Because we invested in it for such a long period of time, it also is a little bit of a disappointment.

Ms. Petralia's remarks show that she prioritized the students' feelings and wanted her students to have a positive experience besides exposure to coding.

Instruction. Although Ms. Petralia needed some support from the researchers in the classroom at the beginning of her very first session, she assumed command of her classroom almost all the time, taking content from the lesson plans and made adaptations whenever she saw fit. A typical 55-minute class would begin with a 10-15 minute "Do NOW!" activity, followed by two or three mini-units, each focusing on teaching the students to program a new device that they could implement on their smart greenhouses, such as the light sensor or the servo, or a new programming concept such as the loop. Standing among the students, Ms. Petralia front-loaded instruction at the beginning of each mini-unit, frequently probing the students with questions, even though her questions tended to go to a few students who always had their hands up. This instruction time was mostly unplugged – the students were not allowed to use their laptops. Following the instruction, students followed step-by-step instructions on what to do on their computer,

while she demonstrated on her computer projected on the screen. She would then give students time to work on their own on their computers, walking around to answer students' questions, as would other adults in the room, until the time for another mini-unit. Figure 10 shows the structure of a typical lesson from Ms. Petralia.

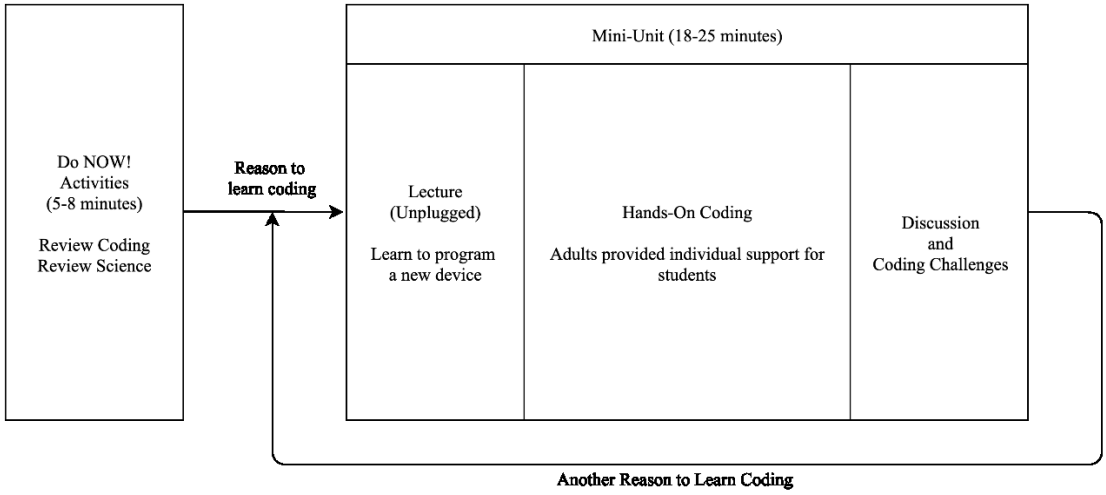

Figure 10. A typical class of Ms. Petralia's

During whole-class instruction, Ms. Petralia used clear and specific language, and students knew what to do at any moment, especially when they were on their computers. The sentences would usually start with "I want you to …" and were repeated adequately so that every student was on the same page. Below is a snippet of her instruction at the end of a session before students left:

> All right, ladies and gents. [whistles to get attention] … What I want you to do is, in your terminal, I'd like you to turn off your grow lights. So gl dot off parentheses. I want you to turn off your grow lights. I want you to exit out of the terminal, you've already saved if you were following directions earlier. Close the laptops. And I'd like you to take apart the LED light, and the sensor, from the MCU board and leave all those things in the box because I don't want you to lose

it. Everything. The bag. Everything in the box. Take it apart with all the pieces in the box…

On the first day of the three-day Design Phase, Ms. Petralia worked with her students to fill the bottoms of the greenhouses with soil and plant their plants. The students designed and built their greenhouses during the next two days, in which she prompted them with questions on the environmental factors that may affect the growth of their plants and the relevant functions of their greenhouses to control these factors. She also provided through Google Classroom pre-written template codes so that once the students determined the optimal growing conditions for their plants, they could alter the code and upload it to the microcontroller without writing code from scratch.

Ms. Petralia came early on the day of the Gallery Walk to help the students set up their greenhouses. The students' smart greenhouses were up and running, glowing in red and blue, streaming data to an online dashboard that she projected for everyone to see, and ready for visitors. The students stood by their greenhouses, answering questions from their curious audience, which included students and teachers from other grades, administrators, parents, and researchers. She was pleased with the outcome of the Smart Greenhouse project and proud of her students for what they had achieved in the past few weeks.

Perceived outcome. With her goal of exposing students to coding and giving them a final product that they could "be proud of," Ms. Petralia considered her Smart Greenhouse project to be a success that exceeded her expectations. At the beginning of the project, she said, "… So if we can get the lights to work and the temperature sensor, that's a win," and after the project, she thought she "met the goals because students were

able to identify on their own, without any assistance, what codes that were needed to turn things on and off." Further, while whether the students mastered the more advanced constructs in a programming language such as the while loops, if-conditionals, and if-conditionals in while-loops were "going to be a case by case situation," she believed all students were able to use Python at the highest abstraction level:

> … But, I think they were all able to import. They were all able to give it a nickname. And especially because I think we reiterated that several times. … I think if you asked all of them, like what are the three lines of code you need for turning on your lights? I think all of them would be successful with that.

Analysis of Ms. Petralia's Instruction

One might describe Ms. Petralia's instruction as "structured." Indeed, just as her carefully written whiteboard announcements with specific instructional goals and agendas indicated, Ms. Petralia taught each lesson with a clear objective, which was connected to her broader understanding of the interplay of computing and science in the learning environment. In the next few sections, I answer RQ1: how did Ms. Petralia implement and reflect on her instruction in this learning environment? To answer this question, I deconstruct how her understanding of the learning environment influenced Ms. Petralia's instructional practices and decision-making during the project.

Understanding of computing and science

Figure 11 summarizes Ms. Petralia's understanding of the reciprocal relationship between computing and science in this learning environment: science provides purposes for computing, and computing provides tools for science. However, this understanding did not take form until the end of the project, and one aspect of this understanding

developed after the other as Ms. Petralia gained experience teaching computing in her science classroom and developed a “bigger picture” perspective of the project.

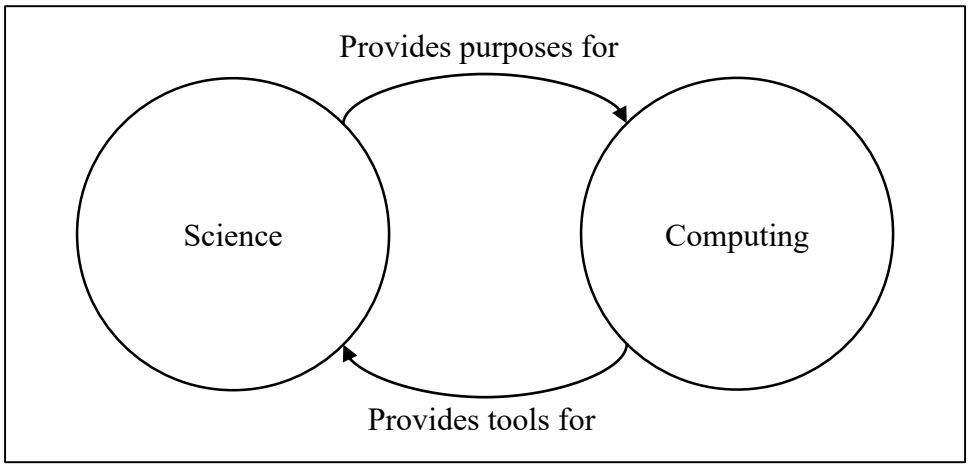

Figure 11. Ms. Petralia's conceptualization of science and computing.

Science provides purposes for computing. From her “50/50” comment on coding and science content “has to be 50/50” and her apparently separate instructional goals for coding at the beginning of the project, it seems that at this moment, she thought she was participating in a coding project where the students were to learn coding and use this knowledge to work on a final product. However, as she used science to make the coding more approachable to the students, whose desire to take care of their plants created an immediate, authentic, and personal need for them to revisit what they had learned previously in school, science then became the constant theme that weaves together the lessons that mainly focused on coding. Having been through in this process, she realized later in the project that science gave coding a purpose so that those who did not already have an interest in coding could find an angle to relate to the coding tasks, which to her, explained the students’ engagement. She reported students having varied responses regarding what their favorite part of the project was, be it playing with LED strips, switching on and off with fans, or even getting their hands in the dirt.

Computing provides tools for science. When asked about what changes that she would make to the lesson plans, Ms. Petralia responded, “I wanted to do a science objective and a coding objective. So, like, ‘Describe the variables that affect plant growth and explain how they use codes to maintain the ideal environment.’ So those are my two objectives for the whole project coming together.” Adding that she had confidence in her students’ abilities to answer that question, she also wished they had more time to collect data and see the effects of different environmental conditions on the plants.

At this moment, although Ms. Petralia still used the term “coding,” her mention of the use of data indicated that her understanding of this learning environment has evolved from doing a “coding” project to doing computing. While the concept of students collecting empirical data using computing devices was noticeably absent from her interview at the beginning of the project, she seemed to have learned more about how computing could be a powerful tool for science. She mentioned thinking about using computing to teach weather, part of the new science standard that she was preparing to teach for the next school year:

> I was thinking of doing, before we do the greenhouses … maybe give [the students] a location where there are certain crops and have them investigate the weather, humidity and air pressure in that area, and then when we come to talk about … why did they not grow lettuce where they were growing … oranges? I don't even know any of this at this point. I've got to do some research myself. Weather is all new to us. It's a new standard that we did not have previously. So, I'd have to kind have to investigate that a little bit. But … if we could figure out specific parts of the world where basil grows really well, and then study the

> weather there or a place where cilantro grows really well, and lettuce grows really well and then talk about that before, they can make the connection …

A powerful point she made on computing was that it could be "meshed" with current and new science standards. That was not part of the Smart Greenhouse project, but using the weather as an example, she foresaw the future of using computing to give students hands-on experience with the science that they are learning.

Cross-curricular Connections. With an improved understanding of computing and boosted confidence about her abilities to teach, Ms. Petralia provided powerful insights into how she saw the opportunity to use computing across content areas. First, she made connections between natural languages and TPLs. For example, she found many similarities between Python and English: one has to be clear, accurate, and specific in both Python and English, and capitalization, punctuation, and conventions are important in both Python and English. Since both languages express meanings, the students could benefit from practicing translation between natural languages and programming languages, which was already a practice that had seen some popularity in communities of programmers. She believed that these commonalities between Python and English could be especially meaningful for English Language Learners (ELLs), if they could incorporate elements from their native languages to write codes that were more relevant to them, such as using Spanish for variable names and even Spanish names for sensors.

She also saw potential opportunities to make computing more culturally relevant in teaching social studies. Recounting her experience teaching at another school, she suggested that "kids go crazy" doing activities relevant to their cultures. Students could

build and code their greenhouses as if they were to grow plants native to their homelands. They could research the climate of the plants' native lands and write essays to persuade local farmers to invest in their greenhouses and grow these plants.

Characteristics of Instructional Practices

In the previous section, I made connections between Ms. Petralia's understanding of the interplay between computing and science and her way to set goals for the project and structure instruction. I now focus on answering RQ1b by describing the main characteristics of the instructional practices she adopted throughout the mini-units to teach coding, beginning with the instructional language that she used.

Characteristic 1: deeply scaffolded language with a focus on meaning. Consistent with the baseline goal of enabling students to "turn things on and off," Ms. Petralia's instruction repeatedly built upon the core unit of Python code to achieve exactly this purpose - the three-step "import-abbreviate-command" procedure reproduced below. Once familiar with these three steps, the students will be able to use the same recipe to write code that drives any other device available. Therefore, Ms. Petralia kept revisiting these steps throughout her instruction.

```
from displays import GrowLight
gl = GrowLight(port=1)
gl.on()
```

In a college-level computer science course, these three lines formally would mean: 1) import into the global namespace a class named 'GrowLight' from an external module called 'displays;' 2) create an instance of this class using its constructor with the 'port' property set to integer '1', assign the instance to a variable called 'tsp;' and 3) call

the ‘on()’ method on the instance to turn on the grow light. However, the following vignette captures how Ms. Petralia taught this import-abbreviate-command procedure to the students for the first time for them to turn on their LED Strips (grow lights):

Vignette 1

Ms. Petralia: So, who has an idea of how we are going to tell Python to bring up the module for our Grow lights? Similar to what we did with math. How do we go about it to import lights?

Student 1: From some library import lights?

Ms. Petralia: Good! So from someplace you are going to import these lights. You have a cheat sheet right on your table. It says on there, from displays import GrowLights. That’s what we call these LEDs. So your first line is going to… Oops. I made a mistake. I can’t get in yet. I forgot. What did I do wrong? [Student 2], what is my first step that I missed here?

Student 2: You missed? Oh Ctrl+D.

Ms. Petralia: I can’t even Ctrl+D yet. What did I forget?

Student 2: Oh Connect.

Ms. Petralia: I didn’t connect. I forgot to connect. The blue button on the top. Press connect. Notice at the bottom. Now it says press Ctrl+D. We need the three arrows in order to get going. Raise your hand if you do not see the three arrows.

Ms. Petralia: So now we are going to import from displays the information for GrowLight. So we are going to type from displays import, and really important that “Grow” is capitalized, and “Light” is capitalized. If you’ve done that correctly, you’ll see three arrows to enter. It’s also on the right side of your cheat

sheet. Maybe you cannot see it the fonts are very small. On the right-hand side of your cheat sheet. You should see three arrows again. Grow Light, capital L. Do it again if something is spelled wrong. From displays… If you do not get three arrows, you get an error sign, you've made a mistake.

Ms. Petralia: So programmers, or coders, they like to have abbreviations, so they don't have to type in the whole GrowLight. Okay, so we are going to name the grow light gl. The lowercase gl, and we are going to tell Python were to find this information. We tell it that we have plugged it into Port 1. So gl is the shortcut for our GrowLight. Captial G, and Capital L. In parentheses, we are going to tell the computer, Python, that the grow light is in port 1. Capital G, capital L, parentheses, port equals 1. Very good. Again, you should see three arrows if you are all set.

[Fire alarm sounds. Instruction Interrupted for approx. 15 minutes]

Ms. Petralia: All right. So, we got a little bit interrupted here, but the last thing that should be on your screen should be, gl equals GrowLight, parentheses port equals 1. Raise your hand if you do not have that. Mine had a little bit of a malfunction here. Can you please then write gl, lowercase gl, and you are going to write a period, on, right, with parentheses. If you do it successfully, your lights should turn on.

[Students' lights came on. Excitement in the classroom]

Compared with the jargon-riddled formal definitions, Ms. Petralia's instructional language was devoid of computer science terminologies and thus much less intimidating to novices. It focused on how each line of code would translate physically to help

accomplish their current task rather than on the abstract computer science constructs. She used various scaffolding strategies to make the explanation as approachable as possible. First, she explained the "import" procedure as telling Python to "bring up" the module for grow lights, which invoked students' prior experience of working with importing mathematical functions such as the square root in a previous lesson. Next, without mentioning any technical details of the second procedure, she provided a simple reason, "So programmers, or coders, they like to have abbreviations, so they don't have to type in the whole GrowLight." Although technically this is only partly true, it established enough background for the students in this context for their tasks at hand, and then she continued, "In parentheses, we are going to tell the computer, Python, that the grow light is in port 1." Finally, because the goal was to turn on the grow light, she said: "Can you please then write gl, lowercase gl, and you are going to write a period, on, right, with parentheses." The code was self-explanatory enough for her not to have to explain what the code meant.

Ms. Petralia adopted similar practices across lessons. Whenever a minimal number of jargons were necessary for her instruction, she provided ample scaffolding for the students. Otherwise she avoided them. For example, when she used the word "module," she compared program modules to "folders" which organize information. Instead of using the word "variable," she scaffolded variable assignments as "giving nicknames." Her focus was more on the underlying meaning of the code than on the mechanics of Python. That does not mean she deemed the mechanics of Python unimportant since she did provide the students with just enough background knowledge to operate on. She considered it more important, however, for the students to understand

at a higher-level what they were instructing the microcontroller to perform with the code and how the microcontroller would respond physically to the code. Vignette 2 is another example in which she taught the concept of a loop. Before this instruction, she had worked with the students to display sensor data on a small OLED screen using the same import-abbreviate-command procedure. However, one issue that they encountered was that the screen would not update periodically, which leads to the need for loops:

Vignette 2

Ms. Petralia: So, there is a problem with this. Can anyone tell me what the problem is? What's the problem with the screen with the usage of the greenhouse?

…

Ms. Petralia: It's not going to update on its own. [It is] just telling you the temperature when you pulled it, right now. So, two hours from now, it's still going to say what it is right now. So our next job is to tell it to update. So we want it to update maybe every 5 seconds… But it's important that it does update so you can see without going back to your computer and checking any code what it is for you to monitor. Okay? And we are going to also code our fan to go on and off when it gets to a certain temperature. We are going to code windows to open and close. … So, in order to make a loop, it's what it's called. We are going to make a loop. We are basically repeating the same instruction for a certain amount of time. So I wanted to update [whistles to get attention] every 5 seconds …

Ms. Petralia: … you are going to type "while True" with a capital T, and you need a colon at the end. Capital T, both of them should be blue. Basically you can use True and False … in Python. … Basically I want you to keep going as long as

this is true. The really important part of this is, after any colon, you need to indent. So this line right here you are going to indent. You need to indent. Any line after while True, it should be indented. Okay? So indent that, and write the next line. Okay? now write that os dot show, sensor, you are basically rewriting the same line, you are just telling it to do it over and again.

The same pattern manifests here again: practical problems drive the need to learn more. The students needed loops so that the screen would update periodically based on the most recent sensor readings, and Ms. Petralia explained it as "repeating the same instruction" without going into much details about it. She prioritized the reasons why they were learning to construct loops to have screen updated repeatedly.

As seen in the underlined portions of the two vignettes, Ms. Petralia's beginner-friendly, meaning-oriented instruction was often accompanied by repetitive descriptions of what exactly to write. She frequently had to spell out the code for the students, word by word, symbol by symbol. Her instructional language did not change even though students became more familiar with the language, which is consistent throughout the Instructional Phase. She had no alternatives but to rely on verbalizing the code since the research team did not present her with more conceptualized language on the precise computer science definitions such as variables, functions, or classes during the PD Phase.

Characteristic 2: novice teaching novices. With the exception of the researchers, both the adults (Ms. Petralia and the paraprofessionals) and the students in the classroom were novice coders in this learning environment. Even though Ms. Petralia had received training, she was still teaching and learning simultaneously – or arguably, learning by teaching. As a novice, Ms. Petralia made mistakes, and it is important to

examine what misconceptions she had and characterize how mistakes from the students were treated in this learning environment to inform future work with both teachers and students.

Mistakes and misconceptions. Ms. Petralia had a few misconceptions due to unfamiliarity with the technology. A recurring misconception was her confusion between the programming language and the development environment of the language. She repeatedly referred to the EsPy Integrated Development Environment (IDE) for MicroPython as "the Python program." At this moment, she was not yet able to distinguish the language from the software program in which she wrote code. She also inaccurately explained that only the "pro" version of the temperature sensor measures temperature and humidity, while the advantage of the "pro" version is precision. Fortunately, these misconceptions were irrelevant to students' understanding of computing and can easily be corrected in the future.

Turning a disadvantage into an advantage. Ms. Petralia did not try to conceal her status as a novice from her students or pretend to be an authority. Only on a few occasions, especially when students asked questions that required knowledge beyond that she was presented with, she would turn to the researchers in the room for support. For example, one student asked about whether the order of function parameters mattered in Python, the answer to which is by no means simple. Sometimes she also gracefully turned her own mistakes into opportunities for teaching. For example, during her demonstration for the students on the grow light in Vignette 1, she accidentally forgot to establish a connection between the microcontroller and the computer, and the code would not work. Having noticed that, instead of simply correcting this problem, she asked a student what

was wrong to show the students what to do under such circumstances. By using these strategies, she never appeared unknowledgeable or unprepared to lose credibility from her students.

Normalizing mistakes. Ms. Petralia leveraged her position as a novice to communicate to her students that it was normal to make mistakes. She said at the end of the project:

> … When I spell something wrong, they were like "Ms. Petralia, you spelled that wrong." And I'm like "I'm a human being! A real boy!" … I'm like "Wow, it shows that I'm human, right?" I can make a mistake too. Just because I'm a teacher does not mean that I'm Miss Perfect here. I'm a horrible speller. But they really think that I should have everything ... Or I miss-say something, I mispronounce or something, … I'm like ... We all have our mistakes; we all have our weaknesses. I don't have to speak perfectly every time I speak…

Ms. Petralia also knew it was essential to recognize that it was acceptable not to understand everything in the first attempt, especially when learning something as challenging as coding. She set this expectation very early on. During "Do NOW!" activities, she would say to her students, "Just to review my expectations, if you don't know some of the answers to these questions, that's totally fine. But what we talk about in class, you should be jotting down whatever you left out."

Learn from mistakes, not to avoid mistakes. One practice that could have happened more often in this learning environment is learning from mistakes. Ms. Petralia was successful in establishing a safe environment for novices to make mistakes, but her instruction did not model what the students should do when they make mistakes so that

they could learn how to independently determine what went wrong and how to make it right. She sought to repeat her detailed, specific instruction on what to capitalize or where not to miss parentheses to prevent the students from making these mistakes. Instead, she could have given the students tools to identify what would happen when they missed parentheses or forgot to capitalize, so they did not have to resort to the adult in the room immediately whenever they made these trivial mistakes.

Characteristic 3: hands-on learning. This section examines the learning activities in which Ms. Petralia engaged her students to scrutinize the role that physical computing devices play and assess the potential of physical computing devices in this type of learning environment.

On the one hand, Ms. Petralia's lessons were hands-on. In each mini-unit, Ms. Petralia left the students with ample time to write code and physically interact with the electronics. For every ten minutes of instruction, students had at least 15 minutes to work in groups on their computer, during which time Ms. Petralia and other adults roamed in the classroom to provide support for students who needed help. For example, after instruction on how to read the temperature and humidity values repeatedly from the temperature sensor, students had opportunities to play with the temperature sensor to change the readings of temperature and humidity. Some would breathe onto the temperature sensor, and some would cover the sensor with their hands. Ms. Petralia also prompted students to try other sources of heat and/or humidity. In a similar lesson on measuring light intensity with light sensors, Ms. Petralia turned switches on and off the lights in the classroom so that the students obtained different light intensity readings on their sensors. It was during this process that Ms. Petralia discovered some students'

misconceptions as to where the sensors were on the PCB boards. This experience gave students a first-hand understanding of what the light intensity unit “lux” meant in real-life light conditions.

On the other hand, the “unplugged” instruction that preceded these hands-on activities were mostly lecture-based. Despite Ms. Petralia’s effort to interact with the students and engage them with probing questions, her instruction, as described in the three vignettes in the previous sections, was teacher-centered and language-oriented. Her PowerPoint presentations were mostly text-based, focusing on the Python codes rather than using visual aids to help students understanding concepts such as loops and modules, which could be attributed to her lack of time to prepare for the lessons and the pressure of finishing the Smart Greenhouse project without ample time. The PD sessions also focused only on familiarizing the teachers with the basics of coding without introducing them to how to teach coding.

Some experimental learning activities designed to integrate computing and science more organically saw some degrees of success. In the same lesson where students learned to program the light sensor, the students used the light sensor to understand the relationship between light intensity and distance. Ms. Petralia instructed them to work in groups, turn on their grow lights, measure the light intensity of red and blue LEDs on the grow lights with light sensors placed at varying distances, and record the result on a worksheet. Then the students graphed the data they collected on the worksheet as homework. In this activity, the students had the opportunity to use the light sensor they had just learned to program as a scientific instrument and perform in data collection and analysis to reach their conclusions. The students practiced the coding skills that they had

just acquired in conjunction with what they already knew in mathematics to engage in scientific practices in NGSS standards.

Ms. Petralia also spent approximately 25 minutes on an experimental scavenger hunt activity near the end of the Instruction Phase. She introduced three battery-powered microcontroller boards that were wirelessly connected to the Internet and streaming data to an online data dashboard, on which three line-charts displayed the real-time temperature, humidity, and light intensity readings from these three sensors. She picked three groups of volunteers, whom she instructed to hide each microcontroller board somewhere in the classroom while the rest of the class were waiting outside the classroom. The task for the rest of the class was to synthesize information from the line charts (Figure 12) on the dashboard to guess where each microcontroller board was. To play this game, students needed to make scientific hypotheses on how their decision on where to place the boards would affect the sensor readings. They also needed to read graphs and combine information from multiple graphs to reach conclusions. Again, all these are important scientific practices recommended within the NGSS framework.

Figure 12. The live data dashboard.

Activities such as these demonstrated the potential of computing devices for integrated learning activities to learn science and computing in meaningful ways. Unfortunately, not much time was spent on discussing and reflecting on what the students did in these activities. Most hands-on learning activities focused on testing whether the physical devices correctly responded to the Python code, and the instruction that led up to these activities, as characterized by the two vignettes in the previous sections, were largely lecture-based, teacher-centered, and language-oriented, somewhat resembling conventional classroom teaching activities.

Challenges

Having elaborated on Ms. Petralia's conceptualization of the learning environment and the characteristics of her instructional practices, I will focus in this section on answering RQ1c and describe the challenges Ms. Petralia encountered while teaching. I structure this section by distinguishing the challenges associated with the characteristics of the learning environment from the practical challenges Ms. Petralia faced in the classroom during instruction. I make this distinction because the former, which are usually uncontrollable practical constraints, can be mitigated with time and experience. Previous sections have already touched upon some of these challenges, and if that is the case, I will refer the reader to the previous sections for more details. The latter, on the other hand, tends to be associated with the design and instructional decisions and should be accounted for in future designs of the learning environment.

Challenges associated with the learning environment.

Lesson planning and logistics. The greatest challenge Ms. Petralia repeatedly highlighted was with lesson planning and logistics. At the beginning of the research

project, the research team did not present to her the exact scope and objectives of the project other than the goal of "building smart greenhouses," because the research team had no prior experience to rely upon at the time and therefore had to adjust the lesson plans as the students made progress. The lesson plans were made available to Ms. Petralia one at a time, and consequently, she had difficulties setting progressive learning goals for the students and thus was unable to make as many adjustments to her instruction as she wanted to. She said:

> I feel like it would be a little more helpful if maybe all the lessons were already inputted into... like I know what they're going to be but they're not... all the lesson plans aren't online yet. So that would help me as a teacher see the next step, because I've been tweaking some things.

The same applies to logistics, especially in terms of the materials needed for class. Ms. Petralia did not know when to expect all the materials for the greenhouses to be ready. These issues, however, tend to happen in pilot projects. After participating in the project, Ms. Petralia not only gained a better understanding of the learning environment but was also able to generate new instructional ideas of learning with computing, as indicated in previous sections.

Difficulties with the software and hardware tools. Ms. Petralia did not report issues with the Python TPL other than a few comments on how it can be unforgiving in spelling and syntax. She understood that all TPLs shared this trait and repeatedly emphasized in her instruction the importance of being accurate and specific. The greatest issue that she had was with the software program in which coding was done. As a young language, MicroPython did not have as much support from the open-source software

community or education community as did Scratch or Arduino, and there were no reliable integrated development environments (IDE)s for MicroPython that was comparable to the Arduino IDE to the Arduino platform. None of the open-source options for MicroPython had all key functionalities useful in the context of education such as syntax highlighting, auto-completion, script uploading and downloading, and step-by-step debugging, not to mention the lack of user-friendliness to young learners. The EsPy IDE was the most feature-rich IDE for MicroPython at the time, but it was far from ideal and sometimes became the source of confusion and redundancy during instruction:

Vignette 3

Ms. Petralia: Okay. [Opens EsPy]. We are going to do something that some of you may not be familiar with at this point. Before we were just typing in the terminal. Now I want you to just actually save your codes that you are writing. So I need everyone to follow along with me. So I want you to go up to the File tab, and you'll see new. [Demonstrates on her computer]. Can you please go over to Python, and click that? You'll get a screen that looks like this. You'll have to save. You can call this, um, let's do "light sensor." And why don't you put your initials. so I know whose light sensor this is. So those are my initials. And Save. So go up to file, new, make a new Python, save as .... [walk around and make sure everyone's got it right].

…

Student: Wait. We are going to write in the upper box, right?

Ms. Petralia: Yes. The upper box, the new box that you just opened…

…

> **Ms. Petralia:** … The first thing you need to do before you run this program is connect your MCU board. Some of you do it, some of you have not. So again, you press this connect button at the top. And you should see in the terminal, Ctrl-D or Ctrl-I. You want to get those three arrows to make sure that you are ready to go. Does everyone have three arrows? (walk around to check on students). When writing code in a new tab here, when you save them, in order for them to get the code to go into the terminal, what you need to do is press in the actual box up here, and you'll see a play button right at the top. And basically when you press play here, it copies it into the terminal and sets the commands. so, once you click this box, you should see a green play button. When you press that green play button, you should get an answer in red at the bottom. So how many lux of light do you have in the room right now? Press in the box, Press IN the box, Press in the box, with your typing. Press the green button…

Vignette 3 shows the many steps that Ms. Petralia had to laboriously instruct the students to follow. There was no room for mistakes in this process, which was so essential, that Ms. Petralia interrupted her instruction and walk around the classroom to ensure that every student followed the same steps. She also had to repeat her instruction a few times to focus the students' attention to these key steps.

Challenges associated with instructional practices.

The dichotomy of success and failure. During her instruction, Ms. Petralia found it most challenging to manage the progress across all students. Her experience is probably not uncommon to teachers in most learning environments that involve a large proportion of hands-on coding. In these contexts, the perceived "success" and "failure" become

simplistic and even dichotomous to the students – the code either works or does not work, and there is little room in-between. When physical devices are involved, the perceived "success" and "failure" are not only self-evident to the students themselves but also visible to their peers, which affects the dynamics of the classroom. For example, when Ms. Petralia taught how to switch on the grow lights, some students successfully switched on their lights faster than the others. These became excited, and some of them were seen turning to their peers and checking on their progress. It was obvious in the classroom who were "successful," and those who had not yet had "success" faced increased pressure from their peers to "succeed." This phenomenon happens especially in classrooms where students have heterogeneous prior experience and interest in coding. Ms. Petralia observed, "… there are 25 students and one of me, some of them are already turning their lights on before we've gotten to that step. Some of them are already blinking their lights, they're going way ahead …. And there are some who are struggling to even connect to the computer."

The linearity of progress. This dichotomous manifestation of "success" and "failure" leads to another characteristic that sets apart similar learning environments involving coding from the others – learning progresses linearly, and "successes" build upon previous "successes." If a student is not "successful" at one stage, it would be difficult for him or her to progress beyond that point. The previous section mentioned the steps that the students had to follow to start coding in the EsPy IDE. If a student misses any step, the chances are that they would not be able to begin working on their code.

For Ms. Petralia, whose goal was to ensure the successes of her students, these issues became particularly salient. She adopted two strategies to overcome this challenge:

repetition and individual support. As is seen repeatedly, Ms. Petralia often had to repeat her instruction multiple times to ensure that all students followed the same steps. She also made sure that the adults in the room, including herself, the paraprofessionals, and the researchers, provided sufficient support for the students whenever they asked for support. However, this proved to be unsustainable for her. In her own words, "[the students] yelled my name a ridiculous amount of times until I was frustrated and I wanted to change my name."

Ms. Petralia's Students

In this section I shift focus away from Ms. Petralia and onto her students and analyze how her students navigated this learning environment that blended science with computing under Ms. Petralia's instruction. I first briefly describe the students included in Ms. Petralia's case and then answer each of the two research questions in separate sections.

Background

Table 7

Ms. Petralia's students (names are pseudonyms).

Name	Class	Team Name	Gender	Race/Ethnicity	Comfort	Skill	Interest
Heather	C	Alpha	F	White	Did not self-report		
Simone			F	White			
Billy	B	Beta	M	White	5	2	2
Min[a]			M	Asian/Pacific Islander	5	3	3
Priscilla	A		F	Black/African American	7	2	5
Hallie	A		F	White	5	2	5
Alexis	A		F	Hispanic/Latino	4	1	2
Jessica	B		F	White	6	2	5

[a] was not interviewed due to absence

Table 7 summarizes the eight students selected in this case using the criteria detailed in the Chapter 4. The first two pairs of students were chosen to be observed. Heather and Simone in Team Alpha were a dynamic and talkative duo of girls. Although they chose not to disclose their levels of comfort, skills, and interest in the pre-survey, Ms. Petralia recommended them as a particularly interesting group to observe. Both Billy and Min of Team Beta reported relatively higher levels of comfort with coding, and although Min self-reported to have a higher interest in coding, it was Billy who was more engaged with the activities in class. Four other female students were also included for a broader range of opinions. Girls were over-represented in this group of eight for the research team to learn more about girls' opinions on this project.

Understanding of the Learning Environment

Figure 13 summarizes the students' understandings of the learning environment. In general, students were not only able to articulate what they learned in science and coding separately in this learning environment, but some also made connections that went beyond merely coding and science.

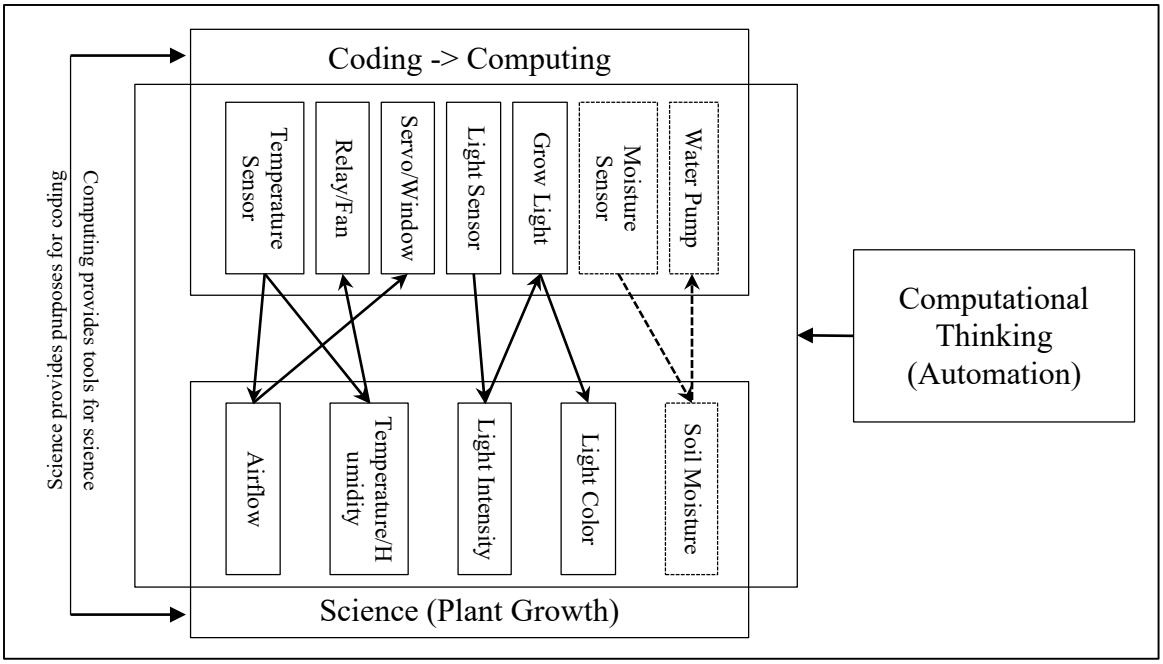

Figure 13. Ms. Patralia's students' understandings and connections

Science. Although the majority of the instructional time were devoted to learning coding, the students were able to enumerate through the environmental variables that affect plant growth in greenhouses, such as temperature, humidity, soil moisture, and light intensity, which were the focus of the science contents. More importantly, they were able to detail exactly how these factors impact the growth of their plants. For example, Billy commented on the respiration of plants:

> I've learned that there's pores on plants that they breathe through and release water from to cool off, and if it's too hot out and not humid enough then water will be drawn out of the plant. And to stop the water being drawn out they'll close the little pores and that can suffocate them because they also breathe through them.

On how light affects plant growth, students indicated that they learned why red and blue lights are helpful for plants. Priscilla indicated that she did not know that artificial lights, in addition to sunlight, could also be beneficial for photosynthesis:

I never actually thought plants can use artificial light to actually work as photosynthesis. Because I always thought it was sunlight. So when we put this on the house, very surprised that this could actually help the plant grow.

Ms. Petralia said that the science of plant growth such as photosynthesis was not new to the students, but the project presented the science to the students in a meaningful way. Comments such as Priscilla's is an indication of how this learning environment engaged the students in authentic scientific practices that helped them apply what they learned in science in the real world, and in this process, they learned more about science than what was presented to them in textbooks, which echoes the emphasis of NGSS (NGSS Lead State, 2013) : "Students cannot comprehend scientific practices, nor fully appreciate the nature of scientific knowledge itself, without directly experiencing those practices for themselves" (p.xv).

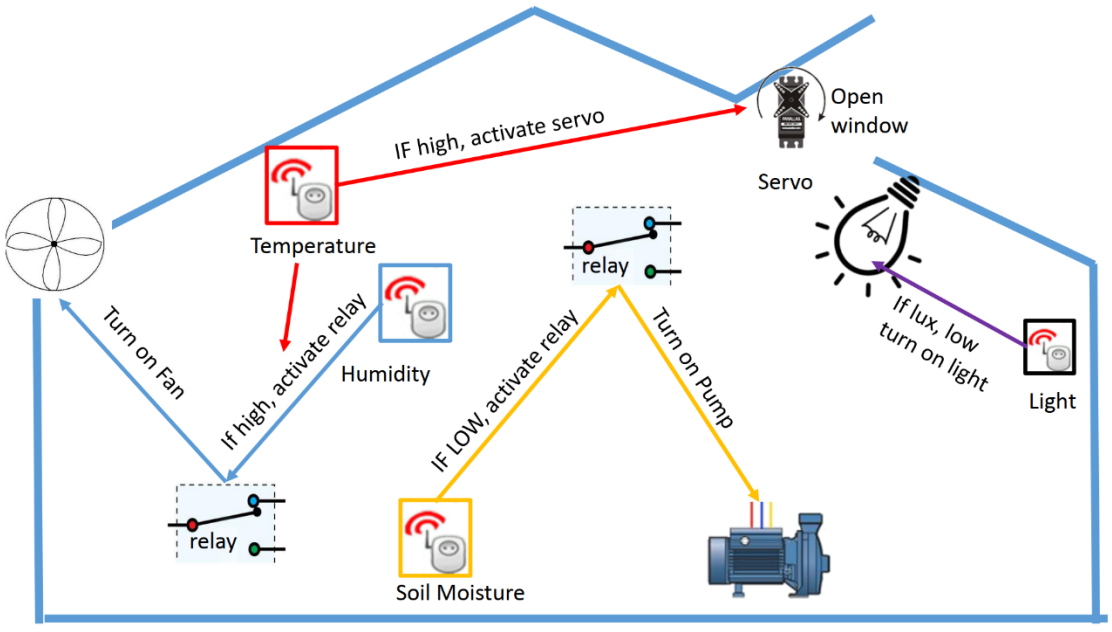

Figure 14. Ms. Petralia's greenhouse automation chart.

Coding. In relation to the science of plant growth, the students were also able to communicate how they applied coding to measure and manipulate the environmental variable and how they envisioned and designed an automated system to maintain optimal environmental conditions in their greenhouses. With the greenhouses in front of them, the students walked their interviewers through the devices that they used for their greenhouses, and how exactly these devices worked together. Since the students' greenhouses were similar, Ms. Petralia made a chart (Figure 14) to visualize for the students how to achieve greenhouse automation with the devices provided, and the students generally described their greenhouses following this chart. They researched the optimal temperatures for the plants they selected (cilantro, basil, or lettuce), and coded the microcontrollers so that if the sensors reported temperatures or humidity over a set threshold, the relay would close the circuit so that the fan turns on, and the servo would rotate so that the arm would turn the specified degrees to lift the window open. Similarly, once the light intensity is below a certain threshold, the grow light would turn on. Although they did not implement the soil moisture sensor, most students expressed that they also wanted to automate watering the plants with a pump or a sprinkler once the soil becomes dry. In addition, Priscilla realized that the grow light also produced heat in the greenhouse:

> … So if the temperature sensor senses that the temperature in there is very hot because of the light shining there, then it will automatically turn it off. I coded that so that happens. Or else, if we left it on and we trusted ourselves to do that, then we will probably forget and kill our plants.

Automation. Building the smart greenhouse engaged the students in the design and implementation of automation with technology. Through this process, they achieved a better understanding of automation not limited to its application in the context of greenhouses. Hallie explained to her family that "we are making [the greenhouse] do stuff for us so we don't have to control easy stuff." Simone imagined, "… I'd have a robot. I'd make a small thing, that you press a button, it would do it all for you. And it'd keep it generating throughout the day, like a portable battery that does it for you." They saw automation as a way of delegating repetitive, mundane tasks to machines in order to save time and improve quality of life, as Simone added, "It'll make your work easier and your day go by quicker when you have things that will do it for you. Instead of you spending hours and hours on time, you could have it start for you."

Priscilla made broader connections between technology, automation, and society. She made connections between the Smart Greenhouse project and what happened in her home recently:

> That's why I think our society and generation is going farther because we're trying to figure out and coming up with ideas to better everyday lives. For instance, someone can have a garden, right? And instead of going on tired of work, if they're using technology? So we're advancing every time more than people in the past.
>
> So I think when I told [my parents] that, I just told them that we're using technology and coding to help our plants. To take care of our plants without having to physically do it every time. And my mom she, her mother, she passed away recently, but her mother used to own a garden. She used to have a garden.

> She's dead so no one takes care of it anymore. But her garden, she used to take care of it. And my uncle who's an engineer he built ... it was outside but he used ... you know how the light over there it's like a pipe thing? And then he just used sprinklers over it to water the plants for her because she was getting old and weak so she couldn't really go outside and do it anymore. So he was using his idea of coding and technology. And he was an engineer so he was very good at combining those things together to help make is mom's garden better.

Not only did Priscilla saw automation as providing convenience, she also saw it as a solution to practical problems, such as world hunger:

> I think if coding becomes a very big thing for people to do, it would definitely help people in other countries. Because they could run out of food and they can't do anything about it because it's the seasons naturally. But if they have something like a greenhouse and be able to code it, it would definitely help them and not have more hunger. It would decrease the hunger levels.

Coding-Science Connection. It seems from the previous section that when asked to specify what they learned about science and coding separately in this learning environment, the students gave mostly similar answers. However, when asked to evaluate what was about this learning environment that was most salient to them, the students had little consensus. In general, between coding and science, some students thought that they learned both, some believed that they participated in a coding project with the goal of building a smart greenhouse, while some others approached the project as designing a smart greenhouse with added coding components. For example, Hallie thought the project was more than coding and science:

> I probably learned a lot of science you know, so that's good. Like the temperatures for it to grow. So that was fun. We learned about what affects a plant that you can control in a greenhouse. We learned about coding because you have to code it. We learned some layout design which was also very good.
>
> In the meantime, Heather said:
>
> I learned that coding isn't just ... I had an image of coding before I started and it was like some weirdo sitting in a room typing on his computer, but it's really not like that. It's so much more different. It's kind of difficult. I knew it was hard to code, but I mean ... I don't know. Sometimes I was like "Wow this is difficult" and other times I was like "Oh I understand this". …
>
> I definitely ... I had no ... When Ms. [Petralia] explained this to us, I had no idea what we were going to be doing. When we started working with the MCU board and stuff and coding it. I was like "Wow, like this is so cool". I didn't think coding had anything to do with plants and keeping them alive. I feel like since I didn't know that coding worked with that I feel like there's a bunch of things that coding can do.

Apparently, in Heather's mind, she understood her experience with the project primarily as learning to code. However, she thought that this experience changed her existing stereotypical beliefs about what coding was and who coders were, which had a somewhat negative undertone from her use of the word "weirdo," as if coding was a mysterious and somewhat an anti-social activity. This image changed after she had concrete experience with physical computing and making connections between coding and growing plants, which demystified coding for her.

Despite also being on Team Alpha, Simone viewed the experience from a different perspective:

> I learned that there's a lot more than just planting a plant. There's more to plant a plant. You need to make sure you have the right temperature, the right humidity control, the battery, you have to make sure everything runs properly to get a full greenhouse working.

To Simone, this project expanded her knowledge about plant growth, showing her the environmental factors the greenhouse needed to control in order for the plants to prosper. Coding was a vehicle through which these variables could be controlled and certain conditions could be maintained.

Comparing Heather and Simone's conceptualizations, it seems that Heather and Simone gravitated towards different elements of the same project. Heather was more excited about coding, and Simone cared more about plants, yet they were both able to invest in some part of the project that mattered more to them and found the other parts to be relevant and meaningful. This essentially mirrors Ms. Petralia's understanding of the interplay of science and computing in this learning environment (Figure 11), where science provides a purpose for computing and computing provides a set of tools for doing science. This feedback loop is especially meaningful in the context of the under-representation of females in today's computer science industry. Stereotypical beliefs about computer programmers such as Heather's discourage girls from engaging in coding and pursuing careers in computer science. This project showed the girls more ways to challenge the stereotypes and engage in the practice of computing.

Computing-science connection. So far, the focus of the discussion in this section has been on "coding." This is because the K-12 students are more familiar with the term "coding" and consequently, "coding" was the term used in all interview questions to the students. The students were, however, also asked to make connections between the use of similar types of "coding" they experienced and real-world work in science, which goes beyond learning to code and examines whether the students thought of the relationship between computing and science.

There was evidence of students thinking of using computational devices to collect information for scientific research. A few believed that sensors could be used to collect information such as temperature and humidity, especially in hard-to-reach places such as a small greenhouse. More students made connections to automation, thinking that scientists could collect information automatically using computational devices without being physically present. For example, Billy said:

> If you're trying to take a bunch of readings from day to day then [the scientists] can't really go out in the middle of the night to take temperature readings. It's for taking measurements or something and they can code the computer to keep taking the measurements for them.

He also reflected on the use of data visualizations like the one he saw in the "scavenger hunt" activity, saying:

> I could use [the data visualization] to change what it does. I could see if there's a weird spike in it or something. I could use that data to fix bugs in it I guess. If something's not working, I could try to change what it does. If the light is flickering, if the light keeps jumping around, I could try to see if I could place it

> in a different spot or change it. Maybe the light could be on the inside I could see how to change it to make the graph more consistent.
>
> I haven't done many tests of it. I have only one test of it. So with a graph it could be constantly analyzing that. We could see more definitively how effective the cooling is. I could also compare inside temperature to the outside temperature. I could have some inside and some outside so I could compare those two to see if inside the greenhouse is actually affecting the growth of the plant.

Billy's comments demonstrated that he was engaged simultaneously in authentic scientific practices and computational thinking. He was not only thinking of using computational devices for automated data collection, but he was also identifying patterns ("a weird spike," "keeps jumping around") from the data collected and using these patterns to reason about the problems that his system might be having and how effective his system was behaving.

Other connections. Besides connections between computing/coding and science, students also made other observations. Simone noted the similarities between writing Python code and writing an essay, stating that the same extent of precision was required for writing in both Python and essay and that it was easier in Python because it provided error messages to help her find the errors. Some students also suggested that this experience gave them a preview of what authentic coding was like, which would be helpful for them to make career decisions. Although some students noted that the smartphone "has coding in it," they tended not to think of the ways coding was used in smartphones, especially in connection to the sensors in them.

Challenges

In this learning environment, the students took a deep dive into computing by taking on the challenge of a TPL without the experience of using BPLs. To echo the "training wheel" metaphor used at the beginning of this dissertation, they learned to ride bicycles without the training wheels. Although it is completely possible to learn cycling this way, one natural question would be on how the experience is. This is what RQ2b focuses on – the experience of the students in this learning environment, especially with respect to their challenges learning a TPL for engaging in scientific practices.

Challenging yet rewarding. In general, most students did report having no experience with such learning and having little knowledge of what to expect from such a project prior to participation. After their participation, they thought the project was "cool" and that they had "fun" from the project, especially in place of a final exam. Even though they found the coding "challenging" and at times "frustrating," they thought this was expected, and the experience of overcoming the challenges became especially meaningful for them. Priscilla summarized her experience most aptly:

> It was kind of stressful, but after you finish it, it's so much like, it's like when you're trying something very hard and it doesn't work. But when it works, you just feel like the happiest person ever.

It seems as though the students did not give up whenever they felt frustrated, as Jessica said, "I would have to take big deep breaths because I am really ... I'm impatient." The reason for this might be that most of them found at least one aspect of the project for themselves to be personally attached to, so not only was overcoming the challenges itself a rewarding experience, the students also learned a life lesson, as Heather said, "I learned

that coding takes a lot of determination and perseverance and optimism because things are going to go wrong a lot."

The inflexibility of TPLs. What exactly caused the initial frustration? It seems that the major source of frustration is with the Python language itself. Most students reported that it was intolerant of mistakes, and therefore, even the smallest typo, such as misspelling a word, forgetting to capitalize, or failing to close a pair of parenthesis, the program would not run. Hallie said:

> It was frustrating because it's very specific and it yelled at me a lot 'cause sometimes I forgot parentheses. But it's okay because we figured it out. But it was fun. It's just like learning another language. So it was fun. I liked it.

She mentioned being "yelled at" because the error messages of Python are meant for programmers, not young children. Nevertheless, most students found the messages useful because the messages helped them identify exactly on which line the error was located, but they also added that this information was not always accurate which also led to frustration. This is a property of the parser of the language rather than a design flaw. Whenever a user forgets to close a pair of parenthesis, the error message will indicate that an error occurred on the next line. Unfortunately, few students recalled seeing any of the friendlier error messages built into the library, probably because those error messages would only appear when the devices were connected to the wrong ports, which must not have happened often because of Ms. Petralia's very specific instruction and the cheat sheet that the students had.

Interestingly, Hallie compared learning Python to learning another human language. Indeed, just like learning another human language, learning Python necessitates

becoming accustomed to its spelling and its syntax rules, but her comment also leaves one wonder if Python's beginner-friendliness also attributed to this connection. Compared with other TPLs such as C/C++ and Java, Python is designed with better readability and similarity to natural languages (van Rossum, 1999), and the design of the library took one step further to emphasize expressiveness and semantic transparency. It is unclear from her comment whether the design played a role, but it does warrant further investigation.

Difficulties recalling code. Another area of difficulty for students was that the they lacked the technical language to describe in detail what they did. For example, some students could not name some of the devices, such as the servo, despite knowing their functions, and most had difficulty describing in detail the code that they wrote to automate the greenhouse, even though they could articulate the functions of the code, as well as the import-abbreviate-command procedure that Ms. Petralia repeated many times. This was not unexpected because of two reasons. First, the importance of technical language was downplayed in instruction by design. While they were exposed to concepts such as loops, students were not formally introduced to the structures of computer science that undergird programming languages, such as variables, data structures, and control flows. Consequently, they might not have developed an internal language that helped them conceptualize their code other than the import-abbreviate-command procedure. Second, due to the limitation of time, the students did not write their code from scratch. They used template code, which they might or might not have fully understood and internalized within a short period of time. The question, again, becomes whether it is necessary for the students to know the technical language and computer science concepts,

especially since their goal of learning in this context is not to learn computer science? I will revisit this question in the Discussion chapter.

Unreliable hardware. In the meantime, some students reported experiencing difficulties with the Wio-Link boards and the sensors used in this project. In this project, we experienced using low-cost open-source hardware to maximize the cost-effectiveness of physical computing to low-income communities. Due to the unavailability of certain devices at the time, the research team sometimes had to hand-make some of the devices, which resulted in some of them being unreliable. In some cases, the students' grow light failed to work properly, and in others, their fans did not turn on. This caused frustration for students because it was difficult for them to identify whether it was their code or the hardware that failed to function properly, which is entirely preventable future iterations.

Summary

In this chapter, I reported the case of Ms. Petralia and her students. Ms. Petralia, a young female science teacher who had initial success in coding and became excited about sharing what she learned with her students, heavily invested in making sure each of her students was successful. She achieved this goal through disciplined instruction and carefully structured lessons with clearly defined objectives. I described how her instruction unfolded based on her conceptualization of the learning environment and how her understanding evolved as she taught and learned. During this process, I highlighted her instructional practices and the challenges she encountered. I also unpacked how her students navigated this learning experience with her and their specific challenges. The next chapter will be a similar report on the case of Mr. Hanrahan and his students.

Chapter 6: The Case of Mr. Hanrahan

In this chapter, I present the case of Mr. Hanrahan and his students using the same structure as Ms. Petralia's case for the organization. The first half focuses on Mr. Hanrahan himself and the other on his students -- each half corresponds with one research question. Since the two teachers essentially implemented the same design of the learning environment, there are inevitably some overlaps. Therefore, assuming that the readers are now familiar with Ms. Petralia's case, I use hers as a basis of comparison for Mr. Hanrahan's case, reporting primarily how his implementation differed from Ms. Petralia's. Whenever overlaps occur, I will refer the reader to matching sections of the previous chapter.

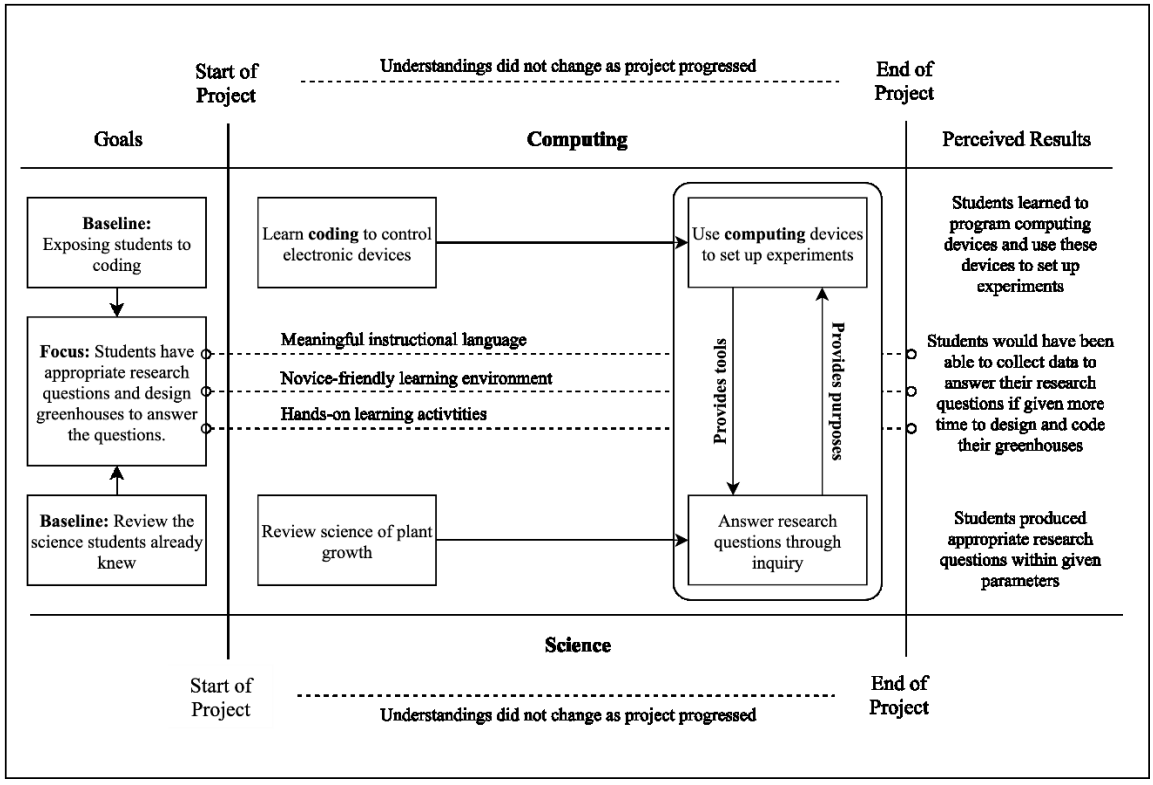

Figure 15. The thematic map for Mr. Hanrahan's case.

Mr. Hanrahan

The purpose of this section is to answer RQ1: How did Mr. Hanrahan implement and reflect on his instruction in this learning environment? Figure 15 maps the main findings of this chapter and how they relate.

Background

Mr. Hanrahan was a veteran science teacher in his 50s at Central Middle School who taught the other half of the Eighth Grade participating in this project. Though white-coded, he self-identified as multi-racial (black and white). His classroom was located away from Ms. Petralia's at the other end of the hallway. Around the whiteboard and on the walls of his classroom were numerous paintings and sports posters. An observer would immediately notice the aquariums at one corner of the classroom in which a few frogs swam idly. On the right side of the whiteboard were columns featuring the lessons of the day and the assignments due. Although Mr. Hanrahan also heavily used the projector during instruction, he would leave a section of the whiteboard empty so that he could write on it. Mr. Hanrahan operated on the same schedule as Ms. Petralia's with his own four blocks of students, but unlike Ms. Petralia, who would distribute the electronics to the students by herself, he would ask the students to retrieve the devices themselves.

Mr. Hanrahan had spent his career teaching science at the middle school level. With his background in biology, he was the more comfortable of the two teachers with the science of plant growth. However, he had no experience with any coding before participating in the Smart Greenhouse project. Nor was he familiar with computational thinking - he took the term "computational thinking" literally and misinterpreted it:

> [Computational thinking] is the idea that there has been some back and forth on the idea of the brain as a computer. … It's gone from rather simplified views where it's simply like data input in, processing, data output. Over the years it's become a lot more complex in all kinds of factors that determine how well things get processed, and the ability of any individual's brain to process and to respond…

His response shows again that while the term "computational thinking" is gaining traction in the education community and becoming more present in science standards, science practitioners remained oblivious to it.

Compared with Ms. Petralia, who had initial success with coding and became enthusiastic about teaching it, Mr. Hanrahan was less comfortable with the coding at the beginning of the PD Phase. Although he eventually caught up with Ms. Petralia's help, his experience was not as smooth as hers. He also spent extra time to practice coding with Python in his own time, but his first attempts were unsuccessful due to a malfunctioning microcontroller board. Due to these initial difficulties, Mr. Hanrahan was reported to be "a little stressed out" about the project, "concerned" about his ability to include coding in his instruction before the start of the project. In his own words, "If any one part of this doesn't go well, then the whole thing is going to blow up and be a nightmare." He also self-reported being "very worried … about the ability of the kids to handle the coding…"

Mr. Hanrahan's Instruction

Goals. At the beginning of the Instruction Phase, Mr. Hanrahan had identical baseline goals as Ms. Petralia's - to expose students to coding so that they have "an option" when making career decisions:

> … the biggest [goal] that at least jumps out at me right away is to hopefully just that the exposure to some programming will ... I know there's a ton of kids who know absolutely nothing about the programming side. For them to see how that works and to potentially then ... Just to know that it's an option. …
>
> In comparison, science had a relatively lower priority. In interview, he said
>
> The fact that [the greenhouse is] going to grow plants is probably not the biggest goal in my mind or even the idea of them understanding what it is that plants need in order to grow. … Most of [the students] probably have a little bit of a sense of that. … It's the coding. It's that this is a way to control things. Maybe someone will end up being a computer programmer.

Focus. Mr. Hanrahan's focus, to the contrary, was very different from his colleague's. He was a verbal advocate of the students having a scientific question:

> I feel like the actual getting the kids to set it up in a way that is relevant and has any purpose at all and any ... Like a focus. I want them to sort of think, and this might be something to really start beating them now with, is that I would tell them, "You need to have a question." Start with a question that you want to answer.…

This focus is much higher-level and more inquiry-oriented than Ms. Petralia's, and Mr. Hanrahan's remark shows that he understood the challenges this might present for himself and his students.

Instruction. It seems that despite his initial struggles with technology, Mr. Hanrahan had a higher expectation for himself and his students. Indeed, Mr. Hanrahan's challenges continued into the Instruction Phase when the researchers taught the first

lesson as he observed. However, he soon started teaching the classes independently, only occasionally asking the researchers to briefly cover more advanced topics that he also wanted to learn more about. Finding his students' engagement and progress "encouraging" after a few sessions, his confidence in himself and his students improved.

Like Ms. Petralia, Mr. Hanrahan also structured his first four lessons after the lesson plans, with an introductory, 10- to 15-minute "Do NOW!" activities to start the lesson, which led to a few 20- to 25-minute mini-units focusing on learning to program a new computing device. With more experience teaching science, Mr. Hanrahan would spend slightly more time at first on teaching science than did Ms. Petralia, getting into more details such as what an LED is. After a few lessons, he realized that students needed more time for coding and skipped the science part of the "Do NOW!" activities entirely.

While sharing a general direction towards which the project was headed, Mr. Hanrahan and Ms. Petralia did not completely agree on the form the final product should assume. Although both supported the idea of students designing greenhouses to answer questions, Ms. Petralia thought that giving students too much freedom to design their final products might be unrealistic given the time constraint. She feared the disappointment the students might feel without a finished final product. Mr. Hanrahan, on the other hand, was more optimistic. The teachers had a heated discussion during a planning meeting, which happened about halfway into the Instruction Phase, where Ms. Petralia became visibly frustrated because Mr. Hanrahan insisted on his plan.

After his fifth lesson Mr. Hanrahan started teaching very differently. The lessons became unstructured and open-ended. He instructed his students to generate a research question that they intended to answer with their smart greenhouses. To ensure that their

questions were simultaneously meaningful and realistically answerable, he made guidelines for students and worked individually with them to provide feedback on their tentative questions. He also worked with groups of students to ensure that their designs were within the parameters that he loosely defined. In other words, Mr. Hanrahan's Design Phase started earlier than planned, so that the students had more time to design and build smart greenhouses that matched their research questions.

Mr. Hanrahan had to miss one day of instruction due to an administrative issue during the Design Phase. Consequently, even though he started this phase early, some students needed time to finish their greenhouses and he was unable to use the day of the Gallery Walk as intended.

Perceived Outcome. Despite experiencing the project much differently from his colleague, Mr. Hanrahan thought the project "went well," especially given the engagement of the students. On the progress of the students, he thought:

> I would say about two-thirds of my kids actually got the greenhouse to work the way it was supposed to properly. So, the fact that there's still a third that really didn't, and it was partly not that I'm Mr. Coding, but the fact that I wasn't here on a Thursday because of a stupid administrative issue, …

The same time constraint also made it impossible for students to collect data and answer research questions:

> It's not enough to see anything, yeah. … [T]hat's probably a reason to lean more towards more what [Ms. Petralia] did then no matter how much we streamline this... there's not going to be enough time for them to really see ... "Wow! Yours versus mine. Look at the difference!" It's just not going to happen.

In terms of baseline goals, he believes that he met both for coding and science. In his opinion, the students learned how coding worked, and even if they would not choose coding as their jobs, they should become more comfortable with the part of their jobs where coding is involved, especially for the female students. He said, "you're making all these young females get exposure to it and saying and realizing, 'This is kind of fun, this is kind of cool, I'm good at this. …'"

Even though what the students learned about science was not part of the science standards, they learned about plant growth:

> I think they learned - they got a better understanding of … the intricacies of that certain types of light and certain intensities of light will increase, allow plants to transition from the growth phase to a fruiting phase. … there are reasons for it….

Analysis of Mr. Hanrahan's Instruction

Compared with Ms. Petralia's instruction with more pronounced structure and detail orientation, Mr. Hanrahan's implementation of the Smart Greenhouse project featured more open-ended scientific inquiry, especially after his early switch into the Design Phase. A veteran teacher, Mr. Hanrahan experienced a different relationship with coding than did Ms. Petralia. Unlike the latter, who became comfortable with coding much faster, he struggled initially but became more confident as he saw the engagement from the students.

Did he also reflect on his instruction differently than his colleague, and might that difference attribute to implementing their instruction? RQ1 is answered in the remainder of this section in three subsections, each focusing on one sub-question of RQ1. For

example, beginning with RQ1a., how did Mr. Hanrahan understand the interplay between computing and science in this learning environment?

Understanding of computing and science

Science provides purposes for computing. Mr. Hanrahan showed his understanding of the interplay of computing and science at the beginning of the project. When talking about his baseline goals, he remarked, “Some actual carrying out of the growing of the plants is one of the [goals]. … It's nice to have a goal. You need to have something that you're designing, setting the coding to do.” At this point, he already understood that science provides a goal for the students to design and code their smart greenhouses, which is consistent with his focus of supporting his students to develop underlying research questions for their final products to answer. After the project, he articulated the same understanding more aptly:

> **Mr. Hanrahan:** And I picture of the whole [project], big picture-wise, as basically it's a coding project with an underlying reason to do it… If you can give them a reason to do the coding, it makes the coding, more relevant, than it makes them- if you just say code something, then it's-
>
> **Interviewer:** Coding for coding's sake.
>
> **Mr. Hanrahan:** So, to have a tangible goal, and target, and purpose, I think it was important, and it made the kids buy in.

Mr. Hanrahan still considered that he experienced a “coding project with an underlying reason to do it.” Science provided “a tangible goal” that made the coding relevant for more students and resulted in better engagement, but the teaching of science was not essential, and the priority should be coding.

Mr. Hanrahan's priority shows in the adjustments he made to his instruction as the project progressed. On Day 2, he began the class with a 13-minute discussion on the science behind "Why are plants green?" which was based on the 2-page reading material assigned to students. Here is part of the discussion:

Vignette 4

Mr. Hanrahan: So, one of the questions on the agenda today is: why are plants green? Last class, nobody remembered this. I was a little disappointed - it goes back a little bit.

[Student answers the question.]

Mr. Hanrahan: Okay, because of chlorophyll. Okay yes, it's because of chlorophyll, but, why green? The thing to think about is why does the … leaves, … when you look at them, why do they appear green to us? We touched on this a while ago,

Student: Because of the sun.

Mr. Hanrahan: Because of the sun, yes, can you elaborate?

…

Mr. Hanrahan: Ok. Pause there. That's one good thing there. Really really good. Thank you. As far as black and white go, if something is black, we've talked about people walking on like a black sidewalk or something, it's really hot. Because black absorbs all of these wavelengths of light. One of these wavelengths of light is energy. If it's black, it absorbs every one of these different wavelengths of light. Absorbing a lot of energy, so it's going to get hot. If something's white, what's actually happening is all of the different wavelengths aren't being

absorbed. They are reflected off of the surface, and they blend together and appear white to us. So black is absorbing all of these different types of energy, white is reflecting all those energies. So, again back to the question, why are plants green?

…

Mr. Hanrahan: Things are the color they appear to be because those colors are reflected off of the substance. So that shirt is red because the wavelengths of red light are bouncing off of that shirt and coming down to my eye. …

Compared with Ms. Petralia's instruction on science, which was brief but served her purpose of giving coding a purpose, Mr. Hanrahan's was much more detailed. It was not only a review of what the students had already learned about photosynthesis, but it also went beyond the middle school science standards, all for introducing why the students needed the grow lights and why they had colors red and blue. On the same day, however, an unexpected fire emergency compelled Mr. Hanrahan's class to evacuate and resulted in 15 minutes of missed instruction time. Mr. Hanrahan realized that if he spent too much time on science, his students might not have enough time to practice coding, so he decided to remove science instruction altogether and leave the readings as an assignment. Below are his Do NOW! questions for one of the subsequent sessions:

1. Which port does Light Sensor go to?
2. Write down the code that would "turn on" the light sensor and display the intensity (lux)?

These are different from Ms. Petralia's Do NOW! questions, which had one reviewing coding and the other discussing science. Science still came up in Mr.

Hanrahan's lessons but mostly before and between mini-units, since Mr. Hanrahan would make connections between the upcoming instruction and the students' final project, but these connections to science were more closely related to the project's final goal of building a "smart" greenhouse:

> Like if you think back to the greenhouse idea, like smart greenhouse, the idea that, "oh you know what, I want my greenhouse fan to turn on, I know, I'll come in to school on Saturday, and I'll walk over, and I'll push the button, and hold it down, so the fan works, Okay?" That's the opposite of a smart greenhouse. Okay? That's not smart at all. Okay? Um, we need something that, again, can do its own thing on its own.

Computing provides tools for science. The role Mr. Hanrahan thought computing played in science differed very subtly from what Ms. Petralia envisioned. While also thinking of computing as a tool for scientific inquiry, his thought was that computing could do more than collecting data;

> … as a data collector, that's part of what you guys [the researchers] are going to be doing and what I'm going to be doing along the way, too, is I want to be able to say, "What happened?" Groups one, two and three, they maxed the temperature at 75. They used these kinds of plants. Now groups four, five, and six, they used the same plants, but they maxed their temperature out at 10 degrees lower. So, what happened? I want to set up some amount of control and variable out of my own personal curiosity as a person who's curious. I want us to be able to see if we can actually quantify the differences that result from changing it just a couple of variables.

His focus seemed to be more on using the smart greenhouses to maintain different growing conditions for plants so that students could collect data to answer research questions. In other words, he envisioned using computing devices to set up and maintain experimental conditions for scientific experiments. To him, computing devices were comparable to lab instruments because of their abilities not only to collect data but also to precisely control and manipulate variables. At this point, although he was still using the term "coding," clearly, he was already thinking in terms of "computing."

The following instruction that he gave to the students when he started the Design Phase early best exemplifies his understanding of the role of computing to science:

> **Mr. Hanrahan:** … that's what we're doing today. We have to … figure out, along with some discussions with me … We need to figure out what is our Greenhouse going to do? And then we can figure out …what we got to put on there. … If you decide you will only want one window to ever be open, and it's going to stay open all the time, then you don't need one of those little servo things open and close [the window] if it's just open all the time. You might say, "well, I want it to open about two windows for eight hours a day," in which case you need two of those little servo arm things that you can connect and program…
>
> …
>
> **Mr. Hanrahan:** So now you might ask, "how does lettuce grow … if I give it blue light?" … How am I going to be able to answer my question? How am I going to know if my lettuce grew well using blue light?
>
> **Student:** you are going to set up your greenhouse so it only has blue lights?
>
> **Mr. Hanrahan:** Okay, but how do I know if that goes well or not?

…

Mr. Hanrahan: … We can compare it with another group and they grew lettuce, but they grew in with red light. Let's say. So that then after like a week or so we can say, "well here's how. Look at my lettuce! I grew with red light, and this compares to their lettuce, and they grew it with blue light, and you see if there's a difference."

In this example, he wanted the students to create a research question, which would guide the designs of the greenhouses. Compared with Ms. Petralia's students, who built similar greenhouses with slight variations, Mr. Hanrahan's students had the freedom to decide which components to include in their final designs. This process resembles engineering design in that the design addresses a problem, which originates from the research questions that the students wanted to answer.

Reflecting on his attempt to teach the students to set up their greenhouses this way, Mr. Hanrahan made this comparison:

… You really can't do them both. Although, you could kind of blend them. In other words, yes, you want your plants to grow as well as possible, so you focus on that. And, you can still set up a situation where you're comparing data and ... That's the other piece. The way this [project] times out, they're never going to get data. They're never going to be able to really answer their questions.

He saw two potential uses for the smart greenhouses, one was to support optimal plant growth, and the other was to set up experimental conditions to compare data. The former seemed to be Mr. Petralia's focus, and the latter was his. Although he did not

think the two focuses were incompatible, he thought the latter required more time for any measurable differences to manifest themselves.

Mr. Hanrahan's instructional Practices

This section focuses on Mr. Hanrahan's instructional practices – the strategies and instructional decisions that he made to achieve his instructional goals. Unlike Ms. Petralia's instruction, Mr. Hanrahan's instruction underwent frequent adjustments, but the practices remained strikingly similar to Mr. Petralia's. Similar to the corresponding section in Chapter 5, this section first unpacks the characteristics of Mr. Hanrahan's instructional language and hands-on learning activities and then analyzes how he taught coding as a novice himself.

Characteristic 1. Deeply scaffolded language with a focus on meaning. Like that of Ms. Petralia's, Mr. Hanrahan's instruction also relied heavily on language. Even though PowerPoint slides were used, the slides themselves were language-based with few visual aids. The language he used was scaffolded without many technical jargons. Vignette 5 below shows Mr. Hanrahan teaching in parallel with Ms. Petralia in Vignette 1 on turning on the grow light using the same import-abbreviate-command procedure:

Vignette 5

Mr. Hanrahan: The first thing we need to do is just to take a look at what you have in front of you right now. You should have one of these little boards, which should already be connected to the USB port. It's got a little red light on. You have these LED strips, right? Which port is the LED strip plugged into?

[Fire alarm sounds. Instruction interrupted for approx. 15 minutes]

Mr. Hanrahan: Okay so let's get caught up here. So which port is your LED plugged into? On the count of three. 1, 2, 3. [Students answer together] One is correct. Why don't you label it Port 1 so you can see it? So here it goes, 1, 2, 3, and then 4, 5, 6. I label it. The first thing we need to do is kind of get yourself into a specific [inaudible]. So, the first thing you do is go to this file, and click new, and then go to Python. So here, do you see what I see?

…

Mr. Hanrahan: … Just like when we typed import math, it let us do like square roots and other functions. So now having imported GrowLight, we've accessed all the information that we can then use for the GrowLight. The next step is a combination of nicknames and location. We are going to give it a nickname, it's going to be ... instead of GrowLight every time, we can type gl lowercase, equals. What we are telling it is, what gl means is, that means GrowLight. Just one more time, type Capital Grow, Capital Light, and that's the nickname and the location.

…

Mr. Hanrahan: … Here's the tough one. I'll be so impressed if you know the answer to this one. … What did I just type? What does that mean?

Mr. Hanrahan: This means, well, do something. Look for, ... well, look for is not the best one. We are telling the GrowLight program, look for... we did square root last time, math dot square root. "sqrt?" We don't want the math program this time. We want the Grow Light program to look for, you are going to type blink. Don't hit enter yet. Everybody pause. Open your eyes wide open. If you hit enter right

now, it won't work. You need to add a pair of parentheses. Open and close parentheses. Then hit enter.

[Students' lights came on. Excitement in the classroom]

Vignette 5 shows that Mr. Hanrahan's instruction was very similar to Ms. Petralia's. He avoided using jargons, and whenever necessary, he scaffolded the language. For example, he explained that the abbreviation step is assigning "nicknames" to instances of classes, which he introduced as "So now having imported GrowLight, we've sort of accessed all the information that we can then use for the GrowLight." He also repeatedly made connections to students' prior knowledge about the math module, which they learned from a previous lesson. At the end of the vignette, he used exaggeration to draw students' attention to the common mistake of missing the parentheses for method calls.

Mr. Hanrahan's instruction language also shared the same issues as Ms. Petralia's. The scaffolding did not fade as the project progressed, and as the underlined portions show, there was too much focus on the syntax rules and the exact spelling of the words.

Characteristic 2. Hands-on coding activities. When Mr. Hanrahan finished his lectures, his students would then have hands-on time with the computing devices. Like Ms. Petralia, he would also allocate the majority of class time for students to write code and would continue with the next mini-unit whenever he thought most of the students were successful. Likewise, his definition of "success" was that the students were able to reproduce the code taught in his lectures to operate on the computing devices. For example, he checked whether the students could switch on and off relays to control the fans with a button:

Everybody, button pushed, fan on? Yes? [Pointing to different students] Yes? Yes? Button pushed, fan on?

[Checking with every student to make sure they are on the same page]

Working? Working? Not bad at all! Okay!

He also checked after teaching the use of an if-statement to turn on the fan if the temperature sensor reading was above a threshold:

… And so, hit play, and, assuming that hopefully it will give you a readout. Does it give you a humidity reading? 40? 41.6? So maybe check your spacing? It says here, putting your hand around it or breathe on it, and get the humidity over 60, it should turn on.

[Students work on their code. Some made exaggerated breathing noise.]

I find it hard to believe. So right now, every single person here, their fan worked. And it was off initially, and you breathed on it, and it turned on. Everybody? 100% Success! Almost. That's three. That's very, very good. And again, some of you, your bad breath is going to break the relay. Is anybody's relay melting right now [Jokingly]?

On one occasion, however, the hands-on activity involved more science. Like his colleague, Mr. Hanrahan instructed his students to measure the light intensity of the grow lights with the light sensor placed at different distances from the individual LEDs. The students would then make line charts of the data collected with sensor placed at different distances on a graph paper.

Characteristic 3. Novice teaching novices.

Mistakes and misconceptions. Like Ms. Petralia, Mr. Hanrahan also made a few mistakes and had a few misconceptions about coding in his instructional language. However, these mistakes were, in most cases, not misleading for students. One trivial example is that he would call the Wio-Link microcontroller boards "the motherboard." The following quote shows that he had confusion about tabs and spaces in Python:

> … I was almost like, just hit the tab keys. Don't hit the Tab key. Okay? What I want you to do is … we want it to be indented. I want you to hit the space bar five times. Okay? … But don't hit the tab key, so hit, like, while True, colon, enter, then 1, 2, 3, 4, 5 (making hitting the key gesture) spaces, then type, if b.is underscore, pressed, colon, because here is what it might get interesting. This one needs to be tabbed in again. So, for this one, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 (making hitting the key gesture again), double indented. And, be careful with how many spaces. Literally, if you do 9 spaces instead of 10, probably not going to work …

Unlike PLs such as Java and C++, Python uses indentation to indicate a block of code. One could use tab keys or spaces to indent the code and make a block, but it is not recommended to mix tabs and spaces in the same Python file, which might lead to bugs that are almost impossible to detect even for experienced programmers. Here Mr. Hanrahan, having learned from recent experiences with another class, was trying to enforce the use of spaces instead of tabs for indentation. He was not technically wrong, but community convention is usually four spaces instead of five.

Turning disadvantages into advantages. Mr. Hanrahan's status as a novice learning to code led to different dynamics in his interaction with the students. For example, when he was working with a group of students whose code did not work

because of a capitalization issue, he said to the students: “It's maddening, isn't it? It literally could be as simple as a capital ‘S…’” His recent struggles with coding gave him leverage to empathize with the students, making his classrooms an environment where it was safe to make mistakes.

With his recent learning experience, Mr. Hanrahan also knew when other novices were prone to make mistakes. At the end of Vignette 5, he used exaggeration during his demonstration, “*Don't hit enter yet. Everybody pause. Open your eyes wide open.* If you hit enter right now, it won't work. You need to add a pair of parentheses. Open and close parentheses. Then I hit enter.” He made the same mistake of forgetting parentheses after method calls during PD, which became the emphasis of his instruction.

Focus on avoiding mistakes, not learning from them. By using the above strategies, Mr. Hanrahan seemed to be focusing on preventing the students from making mistakes. The following exchange between him and a researcher in the classroom shows their different approaches to mistakes:

Mr. Hanrahan: [Realizing there is a typo on the slide] That should be a “tsp” by the way.

Researcher: What's going to happen if we just keep the “ts” over there?

Mr. Hanrahan: Hopefully, they wouldn't be getting data right now.

Researcher: Why don't we just try it and see what happens.

While Mr. Hanrahan wanted to model “correctness,” the researcher suggested that there were values in making mistakes as well. In addition to making the classroom a safe environment to make mistakes, Mr. Hanrahan could also have capitalized on the errors that the students made and modeled how to debug when mistakes did happen.

Challenges

RQ1c is answered in this section by describing the challenges Mr. Hanrahan encountered during his instruction. As was mentioned in the previous chapter, the two teachers shared many of the challenges while having their unique ones. Following is the unique challenge Mr. Hanrahan faced which refers to the ones that he shared with Ms. Petralia.

Challenges associated with open-ended inquiry. With many components to manage, Mr. Hanrahan's approach to the Smart Greenhouse project with more elements of open-ended inquiry was more challenging than Ms. Petralia's. Wrestling with many practical constraints, Mr. Hanrahan faced the challenge that charting the scope of inquiry within a few parameters and communicating these parameters to the students required individual discussions with them to ensure that their research questions were empirically answerable with their greenhouse designs. The practical constraints were as follows.

The capability of the tools provided. The computing devices provided to Mr. Hanrahan and his students had a range of functionalities that controlled a few variables, such as air temperature, humidity, airflow, and artificial light. However, some key functionalities, such as irrigation control, were not available, which limited the range of questions students could explore:

> **Mr. Hanrahan:** … There's "Does lettuce need a lot of water to grow well?" That's your question. And so, you start thinking about like, how do I set up my Greenhouse to answer that question? However, if you think about it, … we talked about the fact that there might be an actual device that is a soil moisture sensor that if the soil got too dry, you can tell this other device to turn on and pump

water into your Greenhouse. We don't have that. Okay, so this wouldn't be a good question because we can't really control this very well.

The skill levels of Mr. Hanrahan and his students. Another important parameter that the students had to consider was whether their current skill level with coding matched the research questions that they wanted to answer and whether they could work with Mr. Hanrahan or other adults for guidance if there was indeed a mismatch. Mr. Hanrahan seemed to have overlooked this parameter during his communication with the students,

You might say, “Well, I want [the greenhouse] to open two windows for eight hours a day,” in which case you need two of those little servo arm things that you can connect and program so that during the particular hours you say those windows …

Though seemingly simple to implement, timing control of computing devices was not part of the instruction, and implementing it required a much higher skill level with coding than the students’ current levels, which created confusion for students whose designs included timing control during the Design Phase.

Timing. The experiments involved plants, which means that it would take weeks before any differences to manifest themselves, if they do at all. Mr. Hanrahan realized that the students would not be able to obtain meaningful results for their experiments and asked the students to proceed as if they were going to collect data and compare results, which affected the authenticity of the inquiry:

… Again, part of it is that it comes down to, you need to sort of set up your Greenhouse so that if we had like a month to collect data that you would be able

to answer your question. Then your Greenhouse should provide data, evidence, results. That would hopefully allow you to answer whatever your question is.

Another challenge with timing is the amount of time the students required to complete the project. The limitations in the parameters for the questions meant the possibility of multiple revisions to their questions and designs. Consequently, any changes to the project's schedule could become difficult to accommodate. Mr. Hanrahan mentioned that he had to leave for one day because of an administrative issue, and more students would have completed the project had he not missed that day.

Challenges associated with the learning environment. The following challenges arose because of the design of the learning environment and were thus shared with Ms. Petralia, although they affected Mr. Hanrahan to a different extent because of his instructional decisions.

Lesson planning and logistics. Mr. Hanrahan also repeatedly mentioned that not knowing the entire project beforehand affected his abilities to plan for the entire project. However, since he assumed an open-ended inquiry approach, he was more affected by the "unplannable" in practice - without experience teaching such a non-conventional project, he could not know what to plan for. He did not know what problems to anticipate or how long it would require for the students to finish their projects. As he jokingly said, "I kind of just ... What is it? Flying by the seat of your pants, as they say. Just kind of winging it, and sort of make do, and something goes wrong, I can adjust." His approach required more flexibility with uncertainty, which he seemed to welcome as part of teaching in this project.

Difficulties with software and hardware tools. Like Ms. Petralia, Mr. Hanrahan also experienced difficulties with the EsPy program with which the students wrote MicroPython codes. Vignette 3 in the previous chapter showed the amount of effort Ms. Petralia needed to prepare the students to code in EsPy. Similarly, the following demonstrates the efforts of Mr. Hanrahan:

> **Mr. Hanrahan:** The first thing we need to do is kind of get yourself into a specific script. So, the first thing you do is go to this file, and click new, and then go to Python. So here, do you see what I see? So, every day you are going to create a new file. And the file is going to be your group number dash and then the date using 0 and the month number and the two digits. So, for example, you'd be WB so somebody, say group 5, dash. No slashes, no dashes, just WB5 dash 0529.
>
> **Mr. Hanrahan:** Everybody got that?
>
> [Work with students individually]
>
> **Mr. Hanrahan:** If before you haven't done that, you should click on device, and then port. It doesn't matter what port it is, just make sure your port is connected.
>
> Student: it won't let me click port.
>
> [Work with students individually]

This process also took Mr. Hanrahan approximately five minutes before he could continue with his instruction, even though he did not rely on language as much as Ms. Petralia did. Again, this could have been avoided with a more novice-friendly IDE.

Mr. Hanrahan's Students

The rest of this chapter focuses on Mr. Hanrahan's students and the answer to RQ2: How did the students engage with the coding and science in the learning

environment? As with the corresponding section in the previous chapter, a brief description of the students in Mr. Hanrahan's case is presented first, followed by answers to each of the two sub-RQs in the two sections afterwards.

Background

Table 8 shows the information of the eight students included in Mr. Hanrahan's case. The first four were also selected to be observed during the Design Phase. Although Ellen self-reported a comfort score of 1, her performance during the project demonstrated that she was at least as comfortable with coding as her teammate Layla. Team Y, on the other hand, encountered much more difficulties than did Team X, even though Caleb chose not to report his previous experience with coding. Since Liam and Casey elected not to be audio recorded during their interviews, and Bobby was absent at the time of the interview, five interviews were included in the data analysis for the rest of this section.

Table 8

Mr. Hanrahan's students (names are pseudonyms).

Name	Class	Team	Gender	Race/Ethnicity	Comfort	Skill	Interest
Layla	E	X	F	Hispanic/Latino	6	2	4
Ellen			F	Hispanic/Latino	1	3	5
Caleb	C	Y	M	White	Did not report		
Claire			F	White	5	3	4
Blair	B		F	Hispanic/Latino	4	2	3
Bobby[a]	B		M	Hispanic/Latino	4	3	4
Liam[b]	E		M	Hispanic/Latino	5	2	4
Casey[b]	E		M	White	6	3	4

[a] Was not interviewed due to absence.
[b] Agreed to interview, but preferred not to be audio-recorded

Students' understanding of the learning environment

Figure 16 demonstrates how Mr. Hanrahan's students conceptualized what they did in the learning environment. This figure is very similar to Figure 13 of the previous

chapter. Indeed, the two teachers' students shared similar understandings with a few key differences, and this section focuses on unpacking these similarities and differences.

Science. Like Mr. Petralia's students, Mr. Hanrahan's students were also able to name the variables in a greenhouse that affect plant growth shown in the bottom part of Figure 16. However, Mr. Hanrahan's students talked about their smart greenhouses in relation to a research question. For example, Caleb said:

> So, the greenhouse works with the overall question being ... There was another part with another group. How would cilantro and lettuce work under a purple light and not red and blue, the most common lights. Everything was practically the same with airflow working at the same time as a fan would go through here at about 70 degrees. Airflow would go out at 75 humidity. So, we did have to make sure the airflow and fan weren't on the same side because that could shoot out the other side.

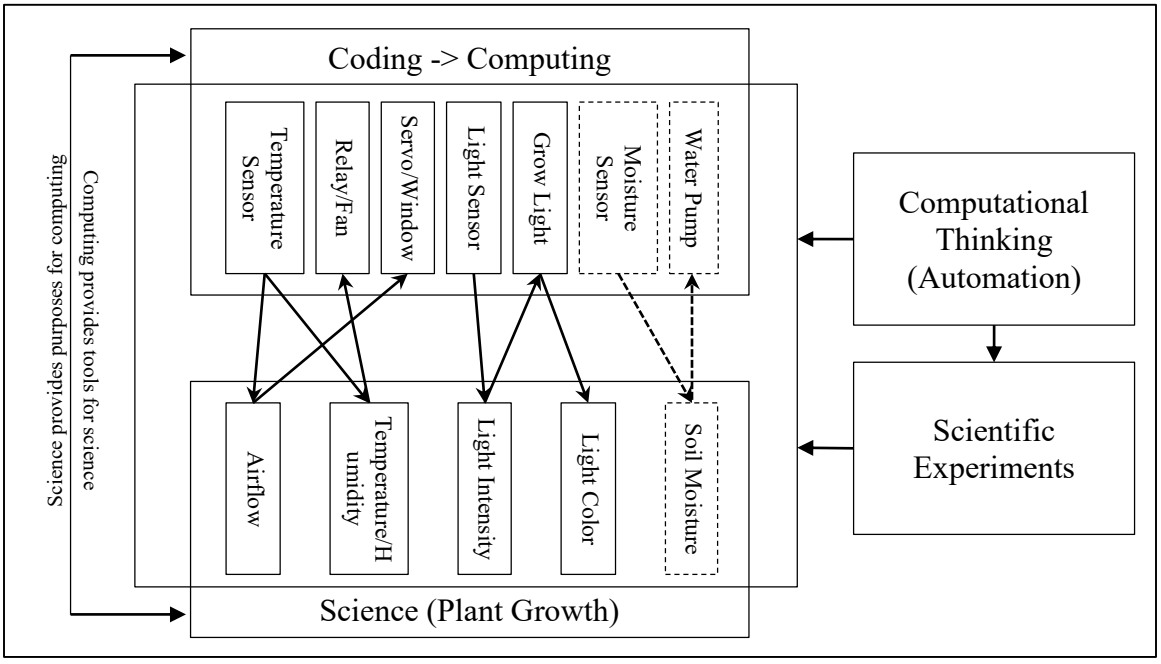

Figure 16. Mr. Hanrahan's students' understanding of computing and science

Caleb mentioned working with another group to establish an experiment with their greenhouses. The goal of the experiment was to test whether cilantro and lettuce would grow better in purple or red and blue lights. He mentioned keeping everything else the same but the colors of the lights. Ellen's group came up with a similar idea:

> **Ellen:** So, blue light and red light are both powerful lights when it comes to growing plants. Purple light is really powerful, too. But we really didn't focus on purple as much, but we just wanted to try out blue and red light because they work good together. So, we wanted to test if they worked separately if they gave the same effect separately or just blue or just red. So, we had the blue light.
>
> **Interviewer:** So, you also did red light in this project?
>
> **Ellen:** No. So, at the beginning, we decided... So, we were going to decide to do red and blue, but since the question was which light... Or a question we had to come up with was: Which light, why you wanted it to work? So, we were like, "Since we can't do that since we can't do purple and blue or red and purple..."

Ellen's comments showed that her group not only considered what they wanted to do but also the parameters within which they asked their question, so that they could design the greenhouse to answer it.

Coding. By requiring students to have overarching research questions, Mr. Hanrahan's instruction had more elements of open-ended inquiry than Ms. Petralia's. However, Mr. Hanrahan's students indicated that they learned to code first and foremost from the Smart Greenhouse project. This understanding is not surprising since Mr. Hanrahan understood the project as a "coding project with an underlying purpose to do it" and phased out science instruction completely in favor of coding instruction. Mr.

Hanrahan's students were not only able to describe the functionalities of their greenhouses the same way Ms. Petralia's students did, but they could also describe what functionalities they decided to include or exclude given their scientific questions. For example, Layla said:

> **Layla:** I used temperature [sensor]. I think that would be enough for it. We said that whenever it would get too hot you can turn off the lights and give it a rest time. I don't think we necessarily needed a humidity sensor. I think a temperature sensor would just do it. We also have this so I can keep it cool, the window.
>
> **Interviewer:** True and you said that you made the window open and close?
>
> **Layla:** Yeah, we did it. I think we are still fooling around with how many degrees it can open up to.

Although Layla had a misconception here (the temperature sensor also measures humidity), her comments show that she experienced the process of engineering design, deciding that given her question, the humidity was not a necessary variable to control. She also used trial and error ("fooling around") to explore the extent she could open the windows with the servo.

Automation. Mr. Hanrahan's students were also able to describe how each component of their greenhouse worked together to achieve automation the same way Ms. Petralia's students did (Figure 16).

Coding-science connection: Although Hanrahan's students indicated that they primarily learned to code from the project, they also welcomed the addition of science. Layla said, "[coding is] definitely [part of science] because [with] coding you can make something like science related like the plants and everything. Maybe engineering, so I

guess it all mixed together." Caleb also commented, "[Science] was fun, coding, too. As well as, messing with plants and technology at the same time, is also really fun." Claire shared similar sentiments:

> [Because] initially when I thought we were doing a greenhouse project, I just thought we were going to put plants in a greenhouse and monitor them every class or something. But when you told us we were coding, I was like, oh that will never work. Now I see how …

Computing-science connection: Like Ms. Petralia's students, Mr. Hanrahan's students understood that they did more than "coding" in this learning environment, even though "coding" was the word they used due to its popularity in K-12 schools. However, the "computing" as they understood seemed slightly different. For example, when asked about the role coding plays in science, Layla answered, "Maybe controlling what [science] does. We're controlling experimenting with the plants, what they do, what they will come out to be. They might die, they might keep growing. So maybe experimenting how they will live." Her focus on controlling experimental condition is best summarized with Claire's answer: "[coding] could help with maintaining the variables of an experiment."

It seems that for Mr. Hanrahan's students, what was most salient to them about computing in science was not only the ability to control and maintain experimental variables but also the precision of the control. Ellen thought, "it's just more organized and more precision like you have very precise things you have to do and the code actually makes it happen." Layla had this unique insight:

> [E]veryday we have to give it water and I keep thinking, oh, it's too much. We do like 150 [ml] every day, and I keep thinking, “Was that too much?” Because I feel that the soil was still a little bit wet. They still hold water in their roots, so I'm thinking, “Oh, is it too much if we keep adding more?” And I think it might affect it in some type of way. I know we have light coming in, but I don't think it's enough. If it's 90 degrees outside you will lose the water and I understand that, but if it's cloudy, maybe the water might not dry up as much.

She was thinking of precision in terms of not only timing but also the quantity of water her system would give to the plants based on other environmental variables such as temperature and light intensity.

Other connections. Mr. Hanrahan’s students also made a few other observations. Asked what language she used to program, Layla answered:

> English, but it's a special kind of English, I guess. English words are included. You can use English words inside the coding. Words like “import” or “display,” stuff like that, but what's also different about it is that you use different numbers and everything.

Ellen compared her past coding experience using a BPL with this current one using a TPL:

> **Ellen:** It's different because you use blocks to make your code and you don't use, on the computer you actually have to type it instead of on the program we used, AppInventor. It's not... You don't type each and everything. You get blocks and you actually make the actual code with the blocks. You don't actually make it typing.

Interviewer: Which do you like better, and which do you think has advantages and disadvantages?

Ellen: I feel like the code for AppInventor is more easier, and I feel like the code that we're doing with the greenhouse is harder because you have to check... You can't make no error mistakes: the spelling, numbers, brackets, parentheses, commas, periods. It's like... You have to be more careful with that. But then in the other one, you make an error and it will tell you your actual error, and it'll just... It's more easier just to use the blocks to make your actual code instead of the typing we had to do for the greenhouse. But, I mean, typing for the greenhouse was really fun, too.

Interviewer: So you actually liked typing it out?

Ellen: Yeah, I actually liked typing it out better than actually using the blocks.

Interviewer: Why did you like it?

Ellen: I felt like it was too easy.

Interviewer: Oh. You liked the challenge?

Ellen: Yeah.

Interviewer: Of the text-based programming language?

Ellen: Yeah. I did.

Challenges

This section focuses on answering RQ2b: what were the students' challenges in this learning environment? I will begin with one challenge that was not exclusive but most salient to Mr. Hanrahan's students. The remaining challenges were shared with Ms. Petralia's students with some aspects more relevant to Mr. Hanrahan's students.

Switching levels of abstraction. The GrowThings library provides easy access to a collection of rich functionalities that are directly related to the construction of greenhouses and classroom teaching activities. By design, the teachers and the students interacted with the high-level functionalities almost exclusively in the learning environment. However, it is when some students wanted to implement functionalities not included in the high-level API that challenges started to emerge. Both Ms. Petralia and Mr. Hanrahan's students experienced this issue to some degree, but it became more pronounced for Mr. Hanrahan's students because these functionalities were crucial to the design of some students' greenhouses to answer their research questions. For example, Blair was set to explore this question: "We were going to make the light so it was two blue lights and one red light, and we were going to make it so my friend could have the opposite, two red lights and one blue light and see if it makes a difference."

However, customizing color patterns on the grow light was not a functionality available within the high-level API. While it was possible to do so through lower-level operations, the code would involve a for-loop with which neither Blair nor Mr. Hanrahan was familiar. Another functionality not yet implemented in the high-level API was timing control of devices, such as toggling the fan on and off in five-hour intervals. Ms. Petralia's students experienced these issues to a lesser extent since these functionalities were not integral to their final products. Billy had a long discussion with a researcher on the precise control of the duration of the grow light, and another researcher worked with Heather and Simone to set their grow lights to a certain color pattern.

Challenging yet rewarding. Compared with Ms. Petralia's students, Mr. Hanrahan's students seemed to have a stronger agreement on their experience with the

project being “hard” yet “fun,” perhaps because their journeys to the final greenhouse design was slightly more tortuous. Lisa’s experience exemplifies their journeys adequately:

> I'm most proud of how we managed to set everything on the greenhouse, how we actually are getting it to work, and how we could assemble everything in place. We had some difficulties during the way, but we managed to get everything in place and where we wanted it to be. I mean, this is like what we imagined in our head what it would be, so it actually came to life. It's a big accomplishment.

Hardware issues. Although Ms. Petralia’s students also experienced difficulties with hardware, Mr. Hanrahan’s students had more, possibly because they had more freedom to include and exclude devices in their greenhouse design. They most frequently reported that it was difficult to memorize the devices and their corresponding ports, even though they had a cheat sheet to refer to when making these decisions. To a lesser degree, the students also felt frustration with unreliable hardware.

Inflexibility of TPLs. The accuracy with which TPLs operate posed challenges for students across the board, as Blair recalled, “I learned that coding is more easier than it actually is. Because the only thing that's hard about coding is doing it specifically how it is. Because if you do a little thing wrong, then it messes up the whole thing.” This sentiment was shared by Claire, who, at the beginning, felt “a bit nervous” because coding “was not a strong suit” of hers. However, she felt after the project, “… after having to copy down a bunch of different codes several times, … I kind of remembered that I had to put parentheses after this and brackets here and comas here and stuff. So, I

kind of remember that now." The students agreed that although the mechanisms of TPLs were initially challenging, they managed to become accustomed in the end.

Difficulties recalling code. Consistent with Ms. Petralia's students, Mr. Hanrahan's students had difficulties describing the code that they wrote to automate their greenhouses. They seemed to remember the import-abbreviate-command procedure but struggled especially when the code incorporated simple loops and if-conditionals that glued together multiple devices. For some students, they also did not recall the precise names of the devices that they worked with, although they did know the precise functions of each computing device.

Summary

This chapter recounts the case of Mr. Hanrahan and his students contrasted with the case of Ms. Petralia and her students. As a veteran teacher who was less comfortable with technology, especially with the coding in this project initially, Mr. Hanrahan focused more on open-ended scientific inquiry and experimental design with the smart greenhouses. His instruction, which was somewhat different from Ms. Petralia's, reflected his understanding of computing and science. Illustrations included above capture his instructional practices and highlight the challenges he encountered. His students also had different experiences with the project and different understandings of the interplay of computing and science in the learning environment in the end. Following in the next chapter is a cross-case analysis summarizing the differences between the two cases.

Chapter 7: Cross-case Analysis and Discussion

In this chapter, I present and interpret the similarities and differences in the findings across the two cases and discuss their implications. The previous chapters presented detailed responses to the two research questions from each case. This chapter discusses what the commonalities across cases mean for future work in designing to implement integrated approaches to computing education at the K-12 level.

The two RQs have both theoretical and practical focuses. The first sub-questions of both RQs intend to explore the connections between computing and science through the lenses of teachers and students who are novices in computing, and that leans toward the theoretical. The remaining sub-questions, on the other hand, focus on the practical – how can we address the practical challenges while adopting the integrated approaches? By comparing and interpreting the findings to these research questions, I address the implications of the findings from both theoretical and practical perspectives, which also leads to a discussion on how we can design learning environments and tools to support the integrated approaches in computing education better.

Computing-Science Connection

Due to the scarcity of theoretical support from the literature of the integrated approach to computing education, this study justified this approach from multiple angles. The beginning chapters addressed the deep historical connections between computing and science, highlighting how they propelled the development of one another and how the multiplying capabilities of modern computers to process large volumes of data resulted in the flourishing of computational sciences. Skills in computing have become a necessity for modern scientists. Computing curricula, however, have just started entering K-12

classrooms. The notion of computational thinking (Wing, 2006, 2008) fueled new waves of thinking in bringing computing into K-12 classrooms by branding "computational thinking" skills as a set of desirable skills for the 21st Century workforce. This movement is hindered partially by the obscurity of the definition of the term, which is still being debated and scrutinized by theorists and experts in computer science (Aho, 2012; Denning, 2007, 2009, 2017a, 2017b, 2017c; Hemmendinger, 2011). To circumvent the definitional difficulties of computational thinking and guide practitioners, computing educators (Barr & Stephenson, 2011; Grover & Pea, 2013, 2017) and organizations (CSTA & ISTE, 2011) alike proposed "practices" of computational thinking, activities in which individuals can engage to develop computational thinking. These computational thinking practices became the entry point of this study. Noting that the set of computational thinking practices and the set of scientific practices from NGSS (NGSS Lead State, 2013) have intersections, I argued that when students are engaged in the shared practices of computational thinking and science, it would be possible for them to develop computational thinking in science classrooms. This argument was then developed into the conceptual framework of this study (Figure 3, Chapter 2).

The first sub-questions of both research questions explore the conceptualizations of the interplay between computing and science from the perspective of teachers and students as they navigated the Smart Greenhouse project. The goal is to examine to what extent empirical data verified the hypothesized conceptual framework. In Chapters 5 and 6, I reported that, despite enacting the same design of the learning environment, the two teachers had subtly different conceptualizations of the interplay between computing and science at the beginning of the Smart Greenhouse project and implemented instructions

differently, which seemed to have influenced their respective students' interpretations of their learning experience and the dynamic between science and computing.

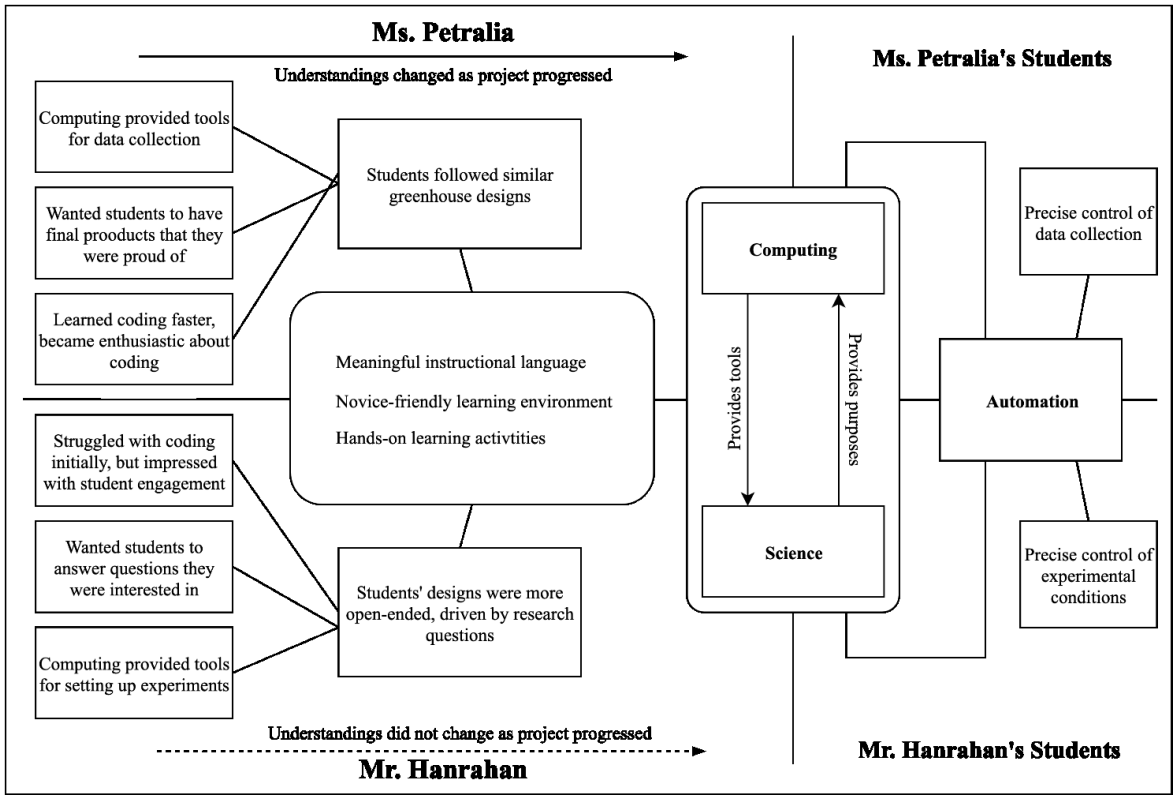

Figure 17. Cross-case themes.

Figure 17 breaks down the similarities and differences between the two cases. It shows that, although the teachers' understandings of the reciprocal between science and computing converged towards the end of the project, their conceptualizations were initially different. Ms. Petralia's understanding underwent an evolution throughout the project, especially in terms of computing. Indicating that she would be re-teaching science "in a meaningful way," she believed that she was teaching coding and did not articulate at the beginning the relationship between science and computing, other than saying that they would be taught "50/50." Having taught in the project, however, it became clear to her that computing could provide powerful tools for science not only in this project but also in her future instruction of new science standards that require

students to collect data with computing tools. She also made cross-curriculum connections between computing and other disciplines such as English and social studies. Science also provided framing and structure for her instruction.

Despite being less comfortable with technology, Mr. Hanrahan had a clear vision of science-computing relationship in this learning environment from the very beginning. This attitude remained constant throughout the project. Mr. Hanrahan indicated his desire for his students to build smart greenhouses with computing devices to answer research questions, which means two things: 1) he believed that science could provide purposes to computing by providing overarching questions; 2) he thought that computing could provide tools to answer these questions. Although Mr. Hanrahan did not make cross-curriculum connections, he did indicate that computing and science gave meaning to each other and thus increased the engagement of his students.

Only a subtle difference existed between how the teachers perceived the role of computing and computing devices in science. Ms. Petralia considered computing devices more as tools for collecting data while Mr. Hanrahan regarded them as tools for setting up experimental conditions for subsequent data collection either manually or automatically.

Ms. Petralia's and Mr. Hanrahan's students' understanding of the connection between computing and science seemed to mirror that of their teachers on a macro and micro scale. On a macro scale, all students, regardless of who their teachers were, agreed that they were engaged in computing to do science, thus echoing the reciprocal relationship that their teachers understood between computing and science. On a micro

level, however, their understanding of the role computing played in science differed subtly, just like their teachers'.

The first difference between the groups lies in what impressed them most about this project and what they have learned above all else. Between computing and science, some students gravitated towards the idea that it was a science project that involved computing, and some thought it was a "coding" project with some elements of science. The former believed that they learned more about the science of plant growth and about using computing devices to measure and manipulate the variables made the learning more meaningful. The latter, however, thought they learned "coding" but in a relatable way. One student mentioned that this experience challenged her stereotypical beliefs about "coders" and "computer programmers" and saw herself engaged in computing. Therefore, with Ms. Petralia's approach, her students saw the project as more versatile and were able to find elements of it to which they could relate.

While a smaller sample size might have played a role, Mr. Hanrahan's students tended to conceptualize the project as a "coding" project, despite their teacher's focus on open-ended scientific inquiry. One contributing factor might be that although Mr. Hanrahan's students were engaged in designing research questions and experiments to answer these questions, they spent the majority of their time to design and code their greenhouses to achieve the experimental conditions, so that they would be able to collect data to answer these questions had there been more time. In other words, the students were engaged in scientific practices, which required extensive coding. Corresponding to this, the science they were practicing also had more elements of establishing scientific experiments when compared with Ms. Petralia's students.

The second difference between the groups lies in what they indicated computing could do for science. Both groups were able to describe with high accuracy the functions of their greenhouses, i.e., what variables the sensors measured and how their greenhouses would respond to changes in these variables. In so doing, the students were using the language of automation, a crucial element of computational thinking across definitions. However, the groups had different foci on the implication of automation in science. Some of Ms. Petralia's students noted that automation provides convenience not only in science but also in everyday life. They stated that scientists could use computing devices to collect data so that they did not have to be present all the time. Likewise, computing can be used in daily life to automate repetitive and tedious tasks to save people's time. Some students also mentioned the precision with which computing devices collect data, especially in hard-to-reach areas. Mr. Hanrahan's students also mentioned precision. However, their focus was on the precision with which computing devices manipulate and control variables to maintain experimental conditions. In that sense, Mr. Hanrahan's students thought of computing devices more as laboratory equipment used to set up experiments while Ms. Petralia's students considered computing devices to be data collection tools.

These differences correspond well with how the teachers intended the students' final products to be. Ms. Petralia wanted her students to build similar greenhouses to support optimal plant growth. Data collection with the greenhouse serves to monitor the conditions inside the greenhouses at all times. Mr. Hanrahan, on the other hand, wanted the students to perform scientific experiments with their smart greenhouses which had the functionality of manipulating and maintaining the conditions within the greenhouses. The

greenhouses would not collect data since no meaningful differences could be observed given the time the students had.

The findings first show the importance, especially in the context of this study, of operationalizing the concept of computational thinking into a set of computational thinking practices (Grover & Pea, 2017). Neither of the science teachers in this study was familiar with the term "computational thinking." Arguably, given the prolonged debates over the definition and scope of computational thinking even among the experts in computer science, as well as the technicalities of terms such as "abstraction" and "pattern recognition," it would be challenging to precisely communicate to science teachers like Ms. Petralia and Mr. Hanrahan, who are not already experts in computing education, the minutia in the term computational thinking.

It is, however, possible to engage the practitioners and their students in the practices of computational thinking without explicit knowledge of computational thinking. As this study shows, having designed and built automated smart greenhouses, the students demonstrated an understanding of automation, a critical element of computational thinking shared by multiple definitions (Lee et al., 2011; Grover & Pea, 2017), beyond the superficial. Not only were they able to articulate that humans could delegate tedious and repetitive work to machines for convenience, some, especially those among Mr. Hanrahan's students, even indicated that machines could achieve levels of precision beyond humans and reduce human errors. The students also showed an emerging understanding of the use of data, which is another major element of the abstraction component of computational thinking. They were beginning to think of using computing devices to design for the collection, analysis, and reasoning of data to answer

research questions in science. These are the essential computational thinking skills deemed useful for everyone, but neither the teachers nor the students necessarily explicitly knew what computational thinking was. They were indeed simultaneously engaged in computational thinking practices (Grover & Pea, 2017) and scientific practices (NGSS Lead States, 2013) to solve real-world problems presented to them, which does seem to verify the conceptual framework.

However, this framework does not fully capture the richness of the data. This study was framed from the perspective of a computer science educator looking to develop computational thinking skills in middle school science classrooms. In other words, the focus was on computational thinking, and the tools designed for this study had the initial purpose of developing coding skills with TPLs in middle school students. However, this focus on computational thinking might be limiting, particularly in this context where multiple competencies in STEM are involved. We saw that, although "coding" sparked widespread interest among the students, not all of them were most excited about or most impressed with this aspect of the project. A sizeable number of students gravitated towards the science of plant growth, and yet some were intrigued by the design of the greenhouse itself. However, it seems that the project saw high levels of engagement not because it catered to this wide range of interest but because it was possible for those who do not identify themselves as "coders" to see how "coding" was integral to what mattered to them. Arguably, Grace would not be alone in having the "weirdo" image imprinted in her mind about programmers. The project demystified coding and broke the stereotypical image of "coders" for them. In that sense, it seems that for integrated approaches,

sometimes it might make more sense for “coding” and “computational thinking” to fade in the background so that the students can choose to learn what is most relevant to them.

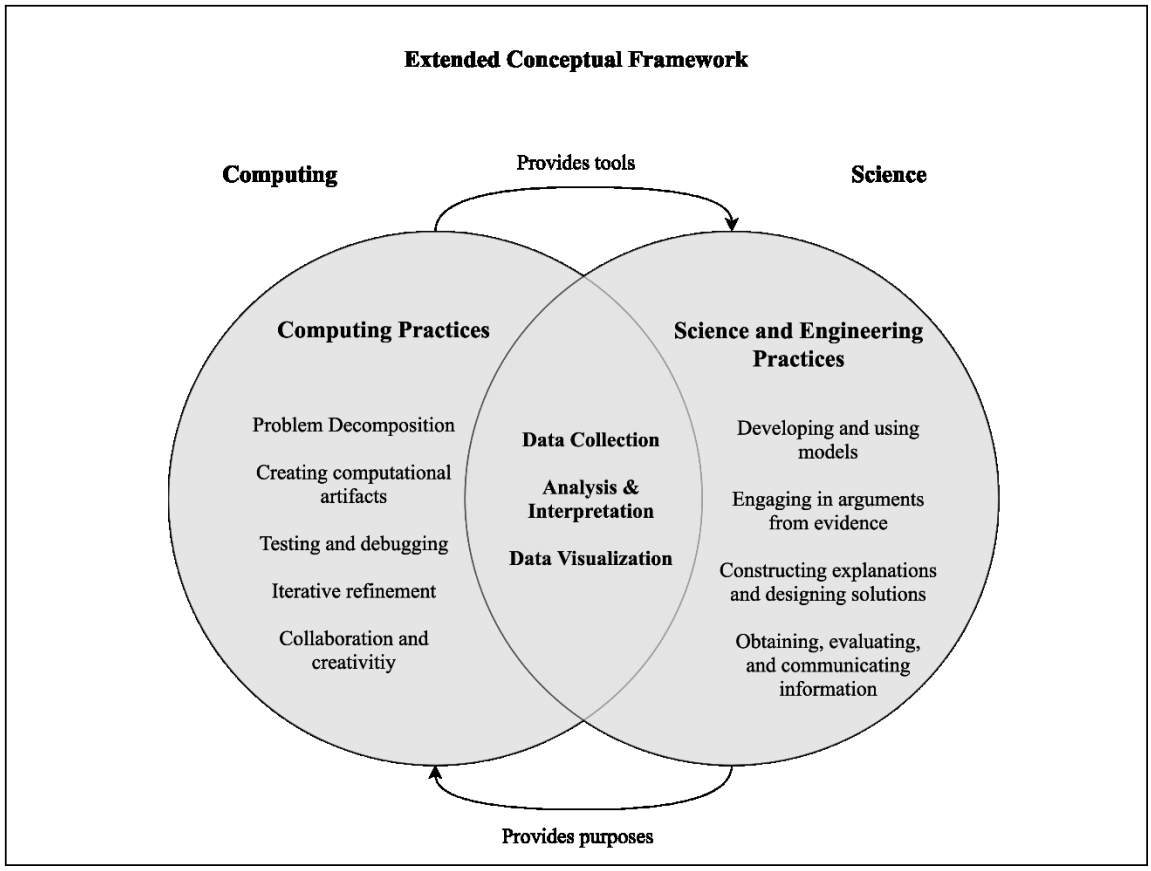

Figure 18. the extended conceptual framework.

To interpret the findings of the case study from a more holistic perspective, we might use the term “computing” in the places of “coding.” The literature (e.g. Grover & Pea, 2017, Wing, 2006) suggests that engaging in coding does not automatically lead to the development of computational thinking, and since computational thinking need not be the only skills that can develop in an integrated context, the overuse of the word “coding” seems a particularly misguided practice. Computing, on the other hand, incorporates not only coding but also practices that do not necessarily involve the programming of a computer, such as using software for data collection, visualization, and analyses. Computational thinking practices fall under this broader umbrella term of computing

practices, but the latter leaves much more room for innovative learning environments where computing is an organic component to the learning of another discipline. Ms. Petralia made many cross-curriculum connections between computing and other disciplines after participating in this project. She saw how the integrated approach could be adopted in the teaching of new curriculum standards in science, such as the weather. She made exciting connections as a novice coder of Python between the programming language and the human language. She also envisioned possibilities of embedding computing in more culturally and linguistically responsive instructional practices. This inter-disciplinary space that involves computing is yet to be explored and awaits much innovation.

Once this focus on coding and computational thinking is removed, we can see a much more symbiotic relationship between computing and science than shared practices (Figure 18). In all participants' conceptualizations of the computing-science relationship, we repeatedly saw this reciprocal pattern emerging: science provides purposes for computing, and computing provides tools for science (see Figure 11, Chapter 5). This pattern not only completes the conceptual framework but also explains why computing and science share practices. Computers were developed to answer questions in other disciplines, especially in science. Although the study of the tools itself has developed into the separate, blooming field of computer science, the direction the field is moving towards is usually propelled by the need from other disciplines. For example, the need for high-performance computing (HPC) in science and other fields is still driving the development of supercomputers today. Thus, an effective way to learn computing might be to learn it in the context it was developed for – problem-solving for other disciplines.

This view naturally separates the teaching of computing from the teaching of computer science. The former focuses on using tools for problem-solving, while the latter focuses on the study of the tools for better problem-solving. While both are worthwhile pursuits, the former might be more practical and relevant for K-12 classrooms and the latter for concentrated studies in higher education.

More importantly, this improved conceptual framework also enables computing educators to see things from science educators' perspectives. The success of the integrated approach requires support from science teachers who allow computing into their classrooms. Science teachers are already inundated with their responsibilities in and out of their classrooms. Before Ms. Petralia and Mr. Hanrahan could participate in the Smart Greenhouse project, they had been burdened with meeting instructional targets and preparing their students for state standardized tests. Changes in standards also mandate their efforts to keep up. Just like Mr. Hanrahan commented, "Usually it's like, 'You should teach A through P. Now you got to teach A through P, but … We're going to add Q, R, and S. … [but] ABC is still there. We're just going to add more.' That just makes it harder and harder to cover everything." Computing could be another non-trivial addition to science teachers' already full plate. On the other hand, science teachers do see the value of computing, especially for their students' future, which is why both Ms. Petralia and Mr. Hanrahan wanted their students to at least have the exposure. If convinced that computing could help their students learn science better instead of being an imposed burden, science teachers could be more willing to introduce computing in their classrooms.

The reciprocal view, coming from science teachers themselves, provides such an angle to solicit their support. The most obvious benefit that science teachers gain, as Ms. Petralia and Mr. Hanrahan reported, is increased levels of interest and engagement of the students. Both recalled students who would typically not be interested in school suddenly showed an interest. Data revealed that the reason for this level of involvement was not because students superficially considered coding to be "fun" or "cool" – they did indeed, but it was more because computing helped frame science as taught in textbooks and tested in exams into a set of authentic, tangible, and empirically answerable questions that the students could explore. These questions were relevant to the students as they were able to make connections to their lives. Then, to answer these questions, the students could use computing tools that they have grown to be comfortable with as "digital natives" in the 21st Century. They also saw the skills that they gain as being valuable and relevant to their future, whether they intended to explore a career in computing or not. To science teachers, by using these tools, their students would also be engaged in scientific practices that new science standards such as NGSS (NGSS Lead States, 2013) are emphasizing on. These, to use Ms. Petralia's words, are presenting science learning "in a meaningful way."

Working with Science Teachers

With the argument made in the previous section, if we can convince science teachers to start embedding computing in their science instruction, the next challenge would be to work with the science teachers to implement this vision. RQ1b specifically explores the instructional practices of the two teachers in order to shed light on how the integrated approach functioned practically with science teachers. Findings to these

questions revealed the constant negotiations of both science teachers between simultaneously being experts in science instruction and novices in computing instruction. As experts, both Ms. Petralia's and Mr. Hanrahan's thorough knowledge of their students became an asset for the research team when the team was co-designing the curriculum with them. They contributed with guidelines on how instructional materials should be presented to the student to maximize usage and comprehension. Even without being briefed on the big-picture ideas and purposes of the research project, they structured and implemented their instruction according to their perception of the project and that steered the project in slightly different directions and resulted in slight differences in their students' learning outcome. Their flexibility with the project's logistics, especially late arrivals of lesson plans, which can be attributed to the exploratory nature of this project, ensured the smooth progression of the project. The project's success owed to a great extent to the expertise and professionalism the two teachers demonstrated.

On the other hand, as complete novices to computing instruction, the teachers were tasked with becoming acquainted with the tools given to them and begin teaching with these tools within weeks. This situation, while challenging, did not place the teachers in a disadvantaged position with the students. In learning to use these tools, the teachers experienced the same triumph and frustration that their students would later also experience. Consequently, the teachers could better empathize with their students, thus creating a safe learning environment for students to experiment and make mistakes and normalize these experiences as part of their learning. For an error-prone endeavor such as learning to code, resilience in not regarding mistakes as failures can be critical to success, and data indicate that the students of both teachers demonstrated such resilience.

However, the teachers did have limited instructional resources on computing to operate on, and both had to rely on what they were presented during PD. As a result, their instructional languages were very similar, and they, for the most part, stuck to the lesson plans provided to them. Both unanimously verbalized their need for being taught as students before they could feel more confident about teaching the students. They also consulted members of the research team in their classroom to provide support, at least at the beginning of the project, although such support was far less needed towards the end. During their coding instruction, they used, somewhat repetitively, similar instructional language, the same language that was used during PD with them before the start of the Instruction Phase. Both teachers also left ample time for students to experience coding by setting standards for success as getting the devices "to work." However, the teachers disagreed strongly on the amount of freedom they should give to the students regarding their final greenhouse designs. The trajectories of the teachers' instruction diverged later with Ms. Petralia's students focusing on similar greenhouse designs while Mr. Hanrahan's students designed different greenhouses to answer the research questions that they were interested in.

These experiences of the teachers are consistent with those of complete novices with computing. The TPACK framework (Mishra & Koehler, 2006, Figure 19) can provide explanations and unpack how we should work with science teachers to implement the integrated approach. Schulman (1986) theorized that teachers not only have pedagogical knowledge (PK) and content knowledge (CK), but they also have pedagogical content knowledge (PCK). Mishra and Koehler (2006) argued that one additional dimension, technological knowledge (TK), teachers' knowledge about specific

technologies, should be included to expand Shulman's framework when discussing technological integration. They proposed that teachers combine their PK and TK to formulate technological pedagogical knowledge (TPK - knowledge about how to integrate technology to implement instruction), and TK and CK to formulate technological content knowledge (TCK - knowledge about how to use technology to present subject matter content). The combination of PCK, TPK, and TCK is called the technological pedagogical content knowledge (TPACK), which refers to teachers' knowledge about the appropriate use of technology to enhance the instruction of content knowledge that maximizes student learning.

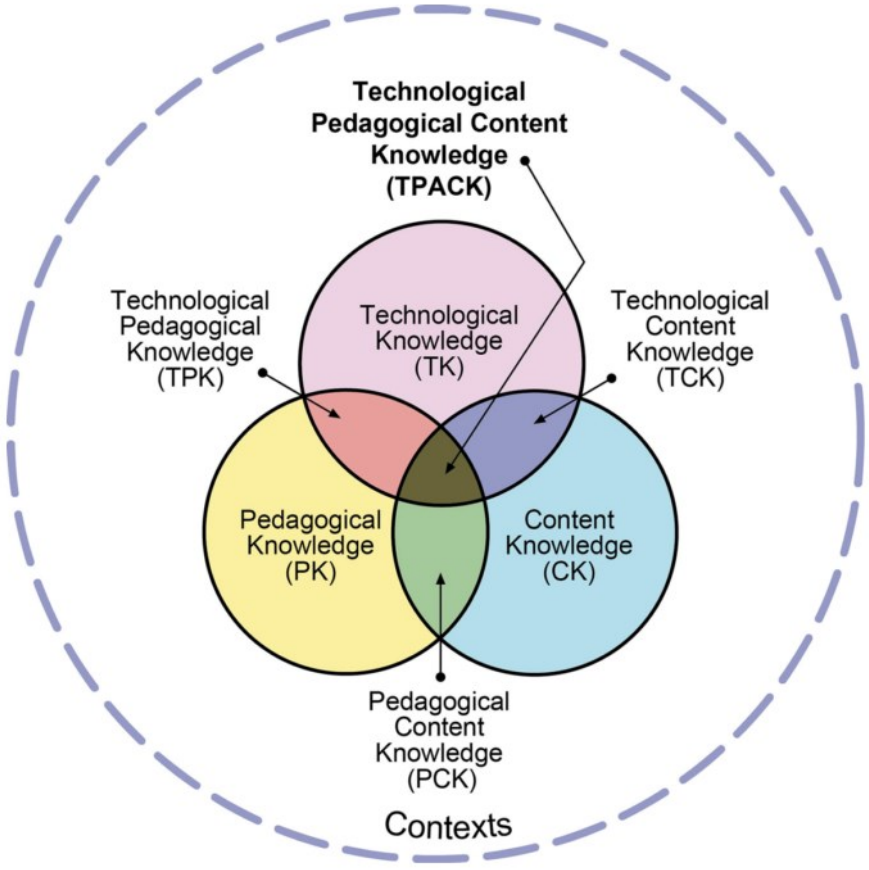

Figure 19. The TPACK Framework (Mishra & Kohler, 2006, p.102)

Applying the TPACK framework to this context, we can deconstruct the double role of the teachers as experts in science instruction and novices in computing as having strong PCK but lacking in TK. Without a background in computing, the teachers, at the beginning of the project, had little resources in TK to draw on to develop TPK, knowledge on developing computing-based activities for science instruction, or TCK, knowledge on using computing for presenting knowledge in science which often requires drawing connections between computing and science. Members from the research team, on the other hand, possessed TK about computing, CK about science, and, arguably, TCK on how computing could be weaved into science content. What they lacked and what the teachers complemented very well, is PK – intimate knowledge about what the students needed and responded well to.

The TPACK framework has illuminated a collaborative model for the science teachers and the researchers from the computing education community to co-design lessons and curricula based on the integrated model. During the PD of this project, the teachers were trained mainly on the use of the computing tools – their TK. However, according to the TPACK model, the research team should have also included in PD, which the two teachers also suggested including in future work with teachers, the "big picture" – the connection between computing and science, the TCK. The teachers, while contributing their PK during co-designing of the curriculum with the researchers, could develop TPK along with the researchers. With their existing PCK, the teachers would be able to weave in their TCK and TPK into TPACK. The last ingredient in this formula is experience. We see that after working with the students in the Smart Greenhouse project, Ms. Petralia started making cross-curriculum connections about how computing could be

embedded in the teaching of new science standards, language, and social sciences. These are emerging signs of the development of her TPK and TCK, as she became more familiar with the technology and acquired TK. For TPACK to develop, she still needs to implement those ideas in her classroom and gain experience from her practices.

The TPACK framework was developed as a model for technological integration into classrooms. The adoption of this model here seems to have steered the discussion from introducing computing into classrooms back to the still-ongoing conversation on how technology should be integrated into K-12 classrooms. These two conversations are not mutually exclusive as we adopt the broader term of computing instead of coding or computer science. Computing is a form of technology, and since computing is viewed in the conceptual framework as providing tools for science according to the new and more comprehensive conceptual framework, it can undoubtedly provide instructional tools for science as well. To introduce computing into science classrooms in such a way is to highlight a less disruptive innovation model than to introduce computer science or computational thinking into the K-12 curriculum. The latter practice fails to consider the myriad challenges the public school systems are facing today and the fact that systemic changes tend not to happen overnight.

Design of the Computing Tools

Another purpose of this research project is to assess the five design principles for extending TPLs for K-12 students with empirical evidence. Doing so with such a pilot project is inherently difficult. To make any causal claims of the impact of these principles requires counterfactuals, e.g., how would the project have eventuated without these design principles in place? Alternatively, one could directly ask the participants to

compare experiences with and without these principles. However, most participants of this project had no similar experiences as bases of comparison. Some design principles, such as scalability and high-ceiling, target impact on implementation and learning in the long run, which this short-term project does not reveal. Therefore, the discussion will have a limited scope on the effectiveness of the design principles and only evaluate whether the goals of the design principles were met in practice. Nevertheless, it is possible to investigate the challenges that the participants experienced to inspect the design and identify areas that need improvement, which is the focus of RQs 1c and 2b.

Shared Challenges for Teachers

Since the teachers enacted the same design of the learning environment, they shared the same challenges that came from the learning environment. The most prominent one occurred with lesson planning and logistics due to the novelty of this learning environment. The lesson plans had to be adjusted according to the students' progress, and consequently, these revisions did not provide the teachers with ample time to prepare lessons. They were also unclear about the overall goals of the project. Because the software and hardware tools used in this project were open-source, some were not fully compatible with educational contexts, and some were not reliable, resulting in difficulties for teachers to use them in class.

Challenges for Ms. Petralia

For Ms. Petralia, whose focus was to ensure the success of all her students, it was precisely the challenge for her. The progression of her classes was linear by nature, meaning that each subsequent lesson had a high dependency on the previous one. If a student missed any one unit, he or she would face mounting difficulties subsequently.

Therefore, Ms. Petralia had to resort to frequent pauses and repetitions to ensure every student was on the same page. In classrooms of approximately 25, this task became challenging for Ms. Petralia.

Challenges for Mr. Hanrahan

Mr. Hanrahan, who adopted a more open-ended approach to his instruction, had fewer issues with maintaining the progress of the students. Since the students could decide the devices to include and exclude in their final greenhouse design, they did not have to know how to program every single device to build their greenhouses. However, Mr. Hanrahan faced the challenge of implementing a more sophisticated approach to this project within multiple practical constraints. Without much preparation, he had to carve out a space in which the students could have the freedom to build their smart greenhouses in order to answer research questions, which were subject to these parameters: the capability of the tools provided, the students' own capabilities to answer these questions, the availability of another group who are interested in similar questions, and the amount of time to finish the final product.

Challenges for the Students

The students from both groups agreed that their experience with the Smart Greenhouse project was "challenging" but "fun." They concurred that they struggled with Python initially, but their comfort improved with experience. While some challenges were also shared across groups, even with different approaches to instruction by their teachers, they affected different groups to varying extents.

TPLs. As novice coders, all students reported frustration with Python's inflexibility, at least at some point. They noted that their programs would not execute

with even the slightest errors in the code, including spelling, capitalization, unmatched parentheses and brackets, and indentation. With the help of system error messages, however, they learned to locate the errors and correct them. The error messages designed into the library, however, were not frequently encountered and thus were less helpful.

Most students also found it challenging to recall what they had written to automate the smart greenhouses. Although most of them were familiar with the import-abbreviate-command procedure repeated many times by the teacher, they had difficulties explaining precisely what they had written to make the greenhouse function as a whole. In other words, although they may have internalized the basics to operate individual computing devices, they still had difficulty managing the more advanced programming structures, such as loops, the if-conditionals, and functions, to manage multiple devices at the same time.

Unreliable tools. The students were provided with open-source devices to build smart greenhouses. These devices, while reducing the costs of engaging students in computing, could be unreliable sometimes, especially since the research team had to assemble some of them to free students from the chore. When hardware failures happened, they would interfere with the students' ability to locate the error.

Switching levels of abstraction. This challenge was more specific to Mr. Hanrahan's students. In their smart greenhouse designs, they would frequently include functions such as having the LED strip display a custom color pattern or toggle the grow lights on or off in given time intervals, so that they could observe the effect of the colors of light or the durations of light exposure on plants. These functionalities were not available yet in the library, and the students would have to write lower-level code that

they were not familiar with to implement them, which became a hurdle when they were engaged in open-ended scientific inquiry.

Based on the challenges experienced, the following sections will briefly reiterate the goal of each design principle and discuss whether these goals were met based on empirical evidence.

Modularity

The goal of the modularity principle is to enable users to use what they have already learned about programming one computing device as a template to reason and predict what they should write to program another device. For example, the above code shows how to program a "LightSensor," which is a sensor, attached to Port 6, to read light intensity. The users should be able to expect that the way to program a "TemperatureSensorPro," which is also a sensor, attached to Port 3, to read temperature should be very similar.

In practice, both Ms. Petralia and Mr. Hanrahan encouraged students to reason in similar ways to the one above when they were learning to program new computing devices. Therefore, the modularity principle accomplished its purpose in practice.

Semantic Transparency

The goal of the semantic transparency principle is to bridge the gap between the commands issued to the computers or computing devices and the functions these commands accomplish. Unlike BPLs, which, with their colorful blocks, reveal to users the actual functions of these commands, such as "switch on the relay" or "rotate the servo to 45-degree position," TPLs in general focus more on the commands issued to the computer. Therefore, in TPLs users usually write "set the output level of Pin 5 to high" so

that the relay switches on, or “set the PWM duty to 52” so that the servo rotates to the 45-degree position. This gap makes it difficult for novices to grasp conceptually the meaning of the code written in TPLs.

The semantic transparency principle focuses on translating what users want to accomplish functionally to sequences of commands that the machines understand. It enables users to write code such as “relay.on()” to switch on the relay and “servo.set_position(45)” to set the servo to the 45-degree position. The users still write with the syntax of the TPL and thus learn the TPL itself, but it frees them from the need to learn the electronic and algorithmic details for the time being so that they can spend more time on other learning tasks. It also frees instructors from having to explain to the students the electronic and algorithmic details, especially if these details are irrelevant to the instructional goals.

In practice, we see evidence that could be connected to the positive impact of this design principle. We see both Ms. Petralia and Mr. Hanrahan frequently using the import-abbreviate-command procedure to teach students how to control computing devices. While the first two steps of the procedure were merely initialization steps, the third one – one that involves code such as “relay.on()” and “servo.set_position(45)” to control computing devices. In the vignettes in Chapters 5 and 6, we see that the teachers seldom saw the need to explain to the students what these codes meant. The teachers were also able to use scaffolded, almost jargon-free instructional language that focused on meaning, that is, the physical manifestation of what the code entailed. As for students, we observe that almost all of them were able to recall the “import-abbreviate-command” procedure. Although reproducing the exact code that they wrote to control the computing devices,

they were mostly able to recall what the code accomplish physically and functionally for their greenhouses.

Although it is difficult to establish that these phenomena directly resulted from the semantic transparency principle, it is reasonable to argue that they are evidence of the teachers' and the students' focus on functions of the code. In that sense, the goal of the semantic transparency principle was accomplished.

The students and teachers, however, did struggle with the syntax rules, frequently making mistakes in capitalization, parentheses, spelling, and indentation. They reported that Python was so specific that even the smallest mistakes would stop the code from being executed. BPLs eliminates the needs of "writing" code and have the mechanisms to prevent mistakes from happening by preventing incompatible blocks from being attached. TPLs, however, do not have such mechanisms, and semantic transparency does not solve this issue. However, instructional approaches can mitigate this issue.

The instruction of the project could be improved by adding more clarity about the exact role each part of the code plays, similar to "parts of speech" in human languages. In the teachers' instructional language, "parts of speech" was ambiguous. For example, in the import-abbreviate-command procedure below, "sensors" is the name of a module, "LightSensor" can be a class or a constructor, "ls" is the name of a variable, "ls.get_lux()" invokes a method on a class instance. The teachers avoided using such jargon most of the time, which was good practice at the beginning. However, they could have, especially in later sessions, gradually established rules, so that instead of saying "write down "LightSensor" with a capital "L" and "S," they could have said, "'LightSensor' is a constructor, so it is always camel-cased." Compared with syntax rules

in human languages, computer language syntax rules are strict with no exceptions and are thus easier to learn. Future designs of instruction should capitalize on the inflexibility of TPL syntax rules.

```
from sensors import LightSensor
ls = LightSensor(6)
ls.get_lux()
```

Fail-Safety

The goal of the fail-safety principle is to provide as much novice-friendly and detailed information as possible, in addition to the error messages of the TPLs themselves, to help the users debug their code. For example, if a user specifies a computing device is connected to a certain port in the "abbreviation" step, then there should be mechanisms in place so that if the said device cannot be found at that port. An error message should remind users of that. It is not technically possible to implement this check for all devices, but it should be in place wherever possible.

In practice, the students did not report seeing many of these error messages. First, they did not differentiate these messages from error messages provided by Python. Second, they much more frequently encountered syntax error messages from Python. Therefore, it is difficult to determine from the evidence of this project whether the goal of this principle was achieved. It is nevertheless good practice in software engineering to properly handle as many errors as possible.

Scalability and High-Ceiling.

These two principles are future-oriented. The research project did not involve students working on other learning projects based on Python, such as CodeCombat and

Turtle. It does not provide direct information on the long-term scalability of the library. However, the cross-curriculum connections Ms. Petralia made and the learning activities she envisioned the students could be engaged in with the tools of the Smart Greenhouse project suggest many possibilities of the same tools being used for other projects, which indicates the high-ceiling property of the tools.

Design changes to the Library

The design of the library contributed partly to the success of the Smart Greenhouse project. It supported approximately 200 students' efforts to code their smart greenhouses for scientific inquiry within a short time. The students did find coding in a TPL challenging due to its rigidity in terms of syntax, but they also expressed that coding with a TPL was not so complex that it was beyond their reach. Some limitations to the design and the functionalities of the library posed some challenges for the students, which I will overcome in future designs of the library. This section will cover updates to the package that have already been implemented or will be implemented in the future.

Adding More High-level Features

The library was designed specifically for the Smart Greenhouse project and supports a set of computing devices used to build smart greenhouses. Even though they can be repurposed into other projects, the functionalities available will be limited. Future versions of the library will support a much wider range of computing devices for more exciting learning activities. Additions to the list of supported devices include ultrasonic distance sensors, water temperature sensors, infrared motion sensors, sound sensors, and carbon-dioxide sensors, which can be useful tools to learning projects in robotics, hydroponics, citizen science, and environmental science. Future additions might include

other computing devices such as accelerometer sensors, gesture sensors, touch sensors, and cameras.

In the meantime, new functionalities have been added to expand the available functions to existing devices and improve the interactivities between computing devices. The LED strip was extremely popular with students, and many students wanted to do more activities with them, such as customizing the color patterns of the lights, design animations and changing the colors of the LEDs according to the output of sensors. They now have the option to engage in these activities easily with the new version of the library without diving into the low-level APIs. Within a few lines of code, they will be able to have their LED lights change color in response to different values environmental variables such as temperature and humidity or to physical events such as motions and noises.

Improve Semantic Transparency

The semantic transparency of the code written with the library can be further improved. For example, every student used the “TemperatureSensorPro” class, which is the only class with a “Pro” suffixed to its name. The “Pro” designation was to distinguish a more accurate and expensive version of the temperature sensor from its less accurate counterpart. In practice, however, this name confused both teachers and students. Ms. Petralia thought only the “pro” version measures humidity, and some students thought that the sensor did not measure humidity. This name is now changed to “TempHumSensor” to indicate that the sensor provides both temperature and humidity readings. This change also prevents students from spelling the word “temperature,” which was frequently a source of spelling errors in practices. “The GrowLight” class is

now named “LEDStrip” to reflect the library’s move towards more general usages beyond the context of building smart greenhouses.

The organizational changes to the library can also improve the semantic transparency of the code. With the additions of new computing devices, the “sensors,” “actuators,” and “displays” (Figure 5, Chapter 3) categorization is no longer accurate in capturing the functionalities of all devices. Some devices were even misclassified. For example, the word “actuator” refers to things that make other things move. Therefore, buttons and buzzers, which were placed under “actuators,” do not belong in this category. The new version organizes the classes into two categories – “input devices” and “output devices,” which correspond to the role of each device in computing. This change will prompt the students to think more about the function of each device in computing in addition to the exact type of device they are.

A Better IDE for Better User Experience

The EsPy program was not a user-friendly IDEs and frequently hindered learning for the teachers and students. Although it became less problematic as students became more familiar with it, the time and effort the teachers had to spend on overcoming its idiosyncrasies in early lessons wasted instruction time. This unnecessary complexity also stood against the design philosophy of the library, which is to remove the initial hurdles for novices to engage in computing, and risked causing frustration for them.

As MicroPython gained popularity and support from the education community, more IDEs have become available that are better suited for middle school students. For example, the Thonny Python IDE (https://thonny.org), a Python IDE targeting beginners, has recently added MicroPython compatibility. With an active community developing it,

this IDE is much more mature and much easier to use. It features many helpful tools for beginners to learn coding, such as variable monitors, simple debuggers, and auto-completion. It also operates on Windows, Macintosh, and Linux computers, including Raspberry Pis, which makes it more compatible with existing computing infrastructure in public schools.

Since public schools such as Central Middle School are more likely to have Chromebooks for students, Chromebook compatibility would be a desirable feature in an IDE. A few online and cloud-based IDEs for MicroPython that are compatible with Chromebooks are under active development at the time of this dissertation, and some of them, such as EMP IDE (http://www.1zlab.com/ide/) are becoming promising alternatives to Thonny Python. Its compatibility with the library warrants further investigation.

Design Changes to the Computing Devices

In practice, the Wio-Link microcontroller board and grove-compatible computing devices successfully eliminated the need for wiring, which allowed students to focus on other more critical tasks. The students did, however, experience difficulties memorizing the ports with which each device was compatible, even though the cheat sheet provided them such information. The underlying issue is that electronic devices use different protocols to communicate with the microcontroller. Some send analog signals, some send digital signals, and some communicate through more complex protocols. On the Wio-Link board, different ports support different means of communication, but all ports look the same, which became the source of confusion.

Labeling or color-coding could be used to indicate different underlying means of communication to mitigate the issue. Many electronic devices designed for education such as LittleBits use color-coding to indicate device compatibility. The practice is also omnipresent in BPLs where different types of blocks are color-coded. Since neither the board nor the grove devices are produced with color-coding, the researchers might need to take on the task by themselves. Another strategy is to teach in limited scope the concept of different communication protocols to the students to help them conceptualize why specific sensors are compatible with specific ports, perhaps using analogies such as USB ports or cellphone charging ports.

Limitations

This dissertation research has met with some methodological and practical constraints which might have implications on the quality and generalizability of the conclusions that it draws.

Use of Secondary Data

The use of secondary data means a lack of control over the data collection process, and as a result, the data collected might not be the most appropriate for the research question. This dissertation research could have benefited from triangulation from different data sources. For example, teachers' instruction and students' group work could have been observed in parallel so better conclusions on students' reactions and responses to instruction could have been drawn.

Sampling and Sample Size

Although the two teachers in this research differed somewhat in age, gender, and experience, they taught in the same school to the same demographic group of students,

which limits the benefits one gains from conducting a multiple case study as opposed to a single case study. Due to limitations in time and resources, the sample sizes of students selected for interviews (eight for each case) were small, and the sample sizes of students selected for observations (four for each case) were smaller. Although a range of criteria was used to select students for the sample, the resulting sample might still not be sufficiently representative of the 200 students of Central Middle School. Future efforts should aim at better generalizability and include different research sites and varying the demographics of students, with much larger sample sizes.

Quality of Data and Missing Data

Because the research team only had a small window to conduct interviews with the students before the students left for summer vacation, the team had to solicit help from researchers from outside the research team to interview the students in parallel. Since some researchers were more familiar with the project than the others, the quality of the interviews was not consistent. Some researchers asked probing questions to elicit more information from students at appropriate times while some did not. Because not all students consented to be filmed, the camera consistently pointed to the teachers during their instruction.

This dissertation project also suffered from the issue of missing data. Due to technological failures, the first day of instruction was not recorded. Unforeseeable circumstances such as students' absences and demands to not be recorded resulted in the loss of four out of the 16 planned interviews, reducing the already small sample sizes. Future projects should form data collection plans more thoroughly before the research project so that the quality of data can be improved.

Timing

Although the research project occurred in an in-school setting, it happened after state standardized testing, when teachers had finished their instructional goals for the semester. Therefore, the applicability of the findings of this research to other in-school settings might be limited.

Chapter 8: Conclusion

The purpose of this research is to evaluate the effectiveness of a learning environment that blends computing and scientific practices and of the principles of extending TPLs for computing in K-12 classrooms. A qualitative, multiple case study was conducted to examine the learning experiences of two groups of novice coders. Each group consisted of one teacher and eight students in the eighth grade of an urban middle school who learned computing and science together for approximately three weeks.

The study answered two main research questions. The first one asked: "How did the teachers implement and reflect on their instruction in this learning environment?" The three sub-questions focused on the two teachers' understandings of the interplay of computing and science in this learning environment, their instructional practices, and the challenges they encountered. Data analysis revealed that both teachers agreed that science provided purposes for computing, and computing provided tools for science in the learning environment. However, they disagreed slightly in conceptualizations of using computing as tools for science and structured their instructions differently with different focuses. Despite these differences, the teachers shared similar instructional practices and emphasis on coding with deeply scaffolded instructional languages and hands-on coding practices in a novice-friendly and error-tolerant learning environment. With different instructional focuses, they experienced different challenges while sharing a few due to the design of the learning environment.

The second research question asked: "How did the students engage with computing and science in the learning environment?" The two sub-questions focused on the students' conceptualizations of computing and science in the learning environment

and their challenges. Data showed that although students gravitated towards computing and science to different extents, they had similar understandings of the interplay of computing and science, which mirrored those of their teachers. They also developed computational thinking through automation, but students of different teachers had slightly different understandings of the use of automation in science. They shared most challenges in the learning environment but were affected to different extents, given the different approaches of their teachers.

The study then discussed the implications of the findings on the design of the learning environment and the principles of extending TPLs for computing in K-12 schools. The project showed promises in engaging all students, including girls, in learning through authentic inquiry with computing tools. However, the study also showed areas that needed improvement, especially in working with K-12 teachers without specific knowledge in computing. Implementing learning environments that involve multiple moving components requires careful collaboration between teachers and researchers for their collective strengths.

While not fully adequate for evaluating the effectiveness of the design principles for extending TPLs for K-12, the study nevertheless compared the empirical evidence against the design purposes of the principles. Based on these comparisons, the study stated the current and future changes to the software package used in this project to improve the usability of the computing tools for both teachers and students.

Recommendations for future research

Despite its limitations, this research imagines what science learning enhanced by computing should look like at the upper K-12 level. It contributes to a field of K-12

computing and computer science education, which is increasingly moving towards blended approaches, with empirical investigations into the intricacies of such approaches and with concrete guidelines for future endeavors. The study makes the following recommendations for future research:

Further disambiguation

Since Wing's (2006, 2008) notion of "computational thinking" fueled a new tide of thinking and efforts on introducing computer science into K-12 schools, the field is becoming cluttered with terminologies that are ambivalent at best, which became confusing for practitioners. The term "computational thinking" itself is still beleaguered by authorities in computer science who question its scope in definition, leaving practitioners struggling to understand its meaning. In the meantime, "coding" seems to have become an all-encompassing term at the K-12 level for all computer science-related activities. However, "coding" has a narrow connotation to only computer programming related activities and undermines the diversity of computing activities that students can engage with. On the other hand, "computing" has the benefit of being broader, and there has been more scrutiny about what counts as practices of computing, but it is not to be confused with "computer science," which refers to the study of computers.

The field needs clear guidelines in distinguishing these terms, and researchers and practitioners should be clear about their instructional goals and the learning activities to achieve these goals. There are benefits in engaging students at the K-12 level in activities in coding, computing, and computer science, but if the goal is, for example, developing computational thinking, then it is not necessary to engage students in coding. More research is needed in tools and activities that engage students in practices of computing

without doing coding, which might be more appropriate for integrating computing into STEM classrooms.

BPLs vs. TPLs.

Similar to the term "coding," BPLs such as Scratch, AppInventor, and Blockly have become equivalent to PLs at the K-12 level. This study shows that there are merits in having TPLs, especially those carefully extended with support for K-12 students, as a transitional step, if not as a parallel option. They could serve as "training wheels" for deeper endeavors of computer programming, software engineering, or computer science. Future research should investigate into these areas: First, how does extended TPLs support the transition between BPLs and TPLs? Since the ceiling effect of BPLs is well-documented, it would be valuable to investigate the best timing and the best way to move students towards TPLs, especially those students who aspire to have careers in computer science. Second, are BPLs better than TPLs adapted for K-12 students as first PLs? Research shows that TPLs such as Python are also effective for young children in elementary schools. It would be interesting to see whether new TPLs designed with novices in mind can take the place of BPLs to become young children's first PLs; thus the need for transition can be avoided. Third, are BPLs effective visual scaffolding tools for TPLs? There might be value in juxtaposing BPLs and TPLs so that BPLs can provide visual scaffolding to the structure and meaning of TPLs. The GrowThings (WioPy) library, for example, is inspired by BPLs, and there can be a one-to-one correspondence between programs "written" in blocks and the code written with the library. The learners could benefit simultaneously from both the lower threshold of BPLs and the higher ceiling of TPLs.

TPLs vs. human languages.

Numerous participants made interesting connections between Python and English. While this research did not pursue further the implication of these connections, future research should investigate the value of drawing on students' existing knowledge in human languages when they are learning PLs. At the very least, students should know that PLs share many similarities with human languages, especially in terms of syntax and semantics. BPLs, on the other hand, are different from human languages. It would be interesting to see what implication of this difference between BPLs and TPLs has on learning to program.

References

Aho, A. V. (2012). Computation and computational thinking. *Computer Journal*, *55*(7), 833–835. https://doi.org/10.1093/comjnl/bxs074.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *Acm Inroads*, *2*(1), 48-54.

Barrow, L. H. (2006). A brief history of inquiry: From Dewey to standards. *Journal of Science Teacher Education*, *17*(3), 265-278.

Berland, L. K., McNeill, K. L., Pelletier, P., & Krajcik, J. (2017). Engaging in argument from evidence. *Helping students make sense of the world using next generation science and engineering practices (229-257). Arlington, VA: National Science Teachers Association. Berland, LK & Reiser, BJ (2009). Making sense of argumentation and explanation*.

Berry, R. Q. R. Q., Bull, G., Browning, C., Thomas, C., Starkweather, K., & Aylor, J. (2010). Preliminary considerations regarding use of digital fabrication to incorporate engineering design principles in elementary mathematics education. *Contemporary Issues in Technology and Teacher Education*, *10*(2), 167-172.

Blikstein, P. (2013). Digital Fabrication and "Making" in Education: The Democratization of Invention. *FabLabs: Of Machines, Makers and Inventors*, 1–21. https://doi.org/10.1080/10749039.2014.939762.

Blikstein, P., & Krannich, D. (2013, June). The makers' movement and FabLabs in education: experiences, technologies, and research. In *Proceedings of the 12th international conference on interaction design and children*(pp. 613-616). ACM.

Blikstein, P., & Wilensky, U. (2009). An atom is known by the company it keeps: A constructionist learning environment for materials science using agent-based modeling. *International Journal of Computers for Mathematical Learning*, *14*(2), 81-119.

Brady, C., Orton, K., Weintrop, D., Anton, G., Rodriguez, S., & Wilensky, U. (2017). All roads lead to computing: Making, participatory simulations, and social computing as pathways to computer science. *IEEE Transactions on Education*, *60*(1), 59-66.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Annual American Educational Research Association Meeting*, Vancouver, BC, Canada, 1–25. https://doi.org/10.1.1.296.6602

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research Month 201X*, *XX*(X), 1–27. https://doi.org/10.3102/0034654317710096

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., ... & Layton, R. (2013). API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*.

Chang, C.-K. (2014). Effects of Using Alice and Scratch in an Introductory Programming Course for Corrective Instruction. *Journal of Educational Computing Research*, *51*(2), 185–204. https://doi.org/10.2190/EC.51.2.c

Carver, J. C., Hong, N. P. C., & Thiruvathukal, G. K. (Eds.). (2016). *Software Engineering for Science*. CRC Press.

Chollet, F. (2017). *Deep learning with python*. Manning Publications Co.

Cooper, S., Dann, W., & Pausch, R. (2000, April). Alice: a 3-D tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges* (Vol. 15, No. 5, pp. 107-116). Consortium for Computing Sciences in Colleges.

Cooper, S., Grover, S., Guzdial, M., & Simon, B. (2014). A future for computing education research. *Communications of the ACM*, *57*(11), 34-36.

Creswell, J. W. (2013). *Qualitative inquiry and research design: Choosing among five tradition.*

Dasgupta, S., & Resnick, M. (2014). Engaging novices in programming, experimenting, and learning with data. *ACM Inroads*, *5*(4), 72-75.

Demir, A., & Abell, S. K. (2010). Views of inquiry: Mismatches between views of science education faculty and students of an alternative certification program. *Journal of Research in Science Teaching*, *47*(6), 716-741.

Dewey, J. (1910). Science as subject-matter and as method. *Science*, *31*(787), 121-127.

Denning, P. J. (2017a). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39.

Denning, P. J. (2017c). Computational Thinking Is Not Necessarily Computational Response. *COMMUNICATIONS OF THE ACM*, *60*(9), 8-8. J. (2017b). Computational design. *Ubiquity*, *2017*(August), 2.

Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM*, *52*(6), 28. https://doi.org/10.1145/1516046.1516054

Denning, P. J. (2007). Computing is a natural science. *Communications of the ACM*, *50*(7), 13-18.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational*

Computing Research, *2*(1), 57-73.

Dougherty, D. (2012). The maker movement. *Innovations: Technology, Governance, Globalization*, *7*(3), 11-14.

Ebrahimi, A. (1994). Novice programmer errors: Language constructs and plan composition. *International Journal of Human Computer Studies*, *41*(4), 457-480.

Etheridge, S., & Rudnitsky, A. (2003). *Guidelines for developing inquiry modules. Introducing Students to How We Know What We Know*, Boston, Allyn and Bacon, 27-50.

George, D. (2017). *MicroPython*. Retrieved from http://micropython.org.

Gershenfeld, N. (2012). How to make almost anything: The digital fabrication revolution. *Foreign Aff.*, *91*, 43.

Goldwasser, M. H., & Letscher, D. (2008). *Object-oriented programming in Python*. Pearson Prentice Hall.

Grandell, L., Peltomäki, M., Back, R. J., & Salakoski, T. (2006, January). Why complicate things? introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*(pp. 71-80). Australian Computer Society, Inc.

Grover, S., & Basu, S. (2017, March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 267-272). ACM.

Grover, S., & Pea, R. (2017). Computational Thinking: A Competency Whose Time Has Come. *Computer Science Education*, 21.

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Gucwa, K. J., & Cheng, H. H. (2014). RoboSim for integrated computing and STEM education. In *American Society for Engineering Education Annual Conference and Exposition* (pp. 1-17).

Guzdial, M. (2008). Education Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25-27.

Guzdial, M. (2012). Why Isn't There More Computer Science in U.S. High Schools? Retrieved September 10, 2018, from https://cacm.acm.org/blogs/blog-cacm/156531-why-isnt-there-more-computer-science-in-u-s-high-schools/fulltext

Halverson, E. R., & Sheridan, K. (2014). The Maker Movement in Education. *Harvard Educational Review*, *84*(4), 495–504. https://doi.org/10.17763/haer.84.4.34j1g68140382063

Hanauer, D. I., Jacobs-Sera, D., Pedulla, M. L., Cresawn, S. G., Hendrix, R. W., & Hatfull, G. F. (2006). Teaching Scientific Inquiry. *Science*, 314(5807), 1880–1881. https://doi.org/10.1126/science.1136796.

Harvey, B., Garcia, D. D., Barnes, T., Titterton, N., Miller, O., Armendariz, D., ... & Paley, J. (2014, March). Snap!(build your own blocks). In *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 749-749). ACM.

Hemmendinger, D. (2010). A plea for modesty. *Acm Inroads*, *1*(2), 4-7.

Hodges, S., Taylor, S., Villar, N., Scott, J., Bial, D., & Fischer, P. T. (2013). Prototyping connected devices for the internet of things. *Computer*, *46*(2), 26–34.

https://doi.org/10.1109/MC.2012.394.

Hughes, J., Gadanidis, G., & Yiu, C. (2017). Digital Making in Elementary Mathematics Education. *Digital Experiences in Mathematics Education*, *3*(2), 139-153.

ISTE, & CSTA. (2011). *Operational definition of computational thinking*. Retrieved from http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf.

Jona, K., Wilensky, U., Trouille, L., Horn, M. S., Orton, K., Weintrop, D., & Beheshti, E. (2014). Embedding computational thinking in science, technology, engineering, and math (CT-STEM). In *future directions in computer science education summit meeting, Orlando, FL*.

Kalelioğlu, F., Gülbahar, Y., & Kukul, V. (2016). A Framework for Computational Thinking Based on a Systematic Research Review. *Baltic J . Modern Computing*, *4*(3), 583–596.

Kölling, M., Brown, N. C., & Altadmri, A. (2017). Frame-based editing. *Journal of Visual Languages and Sentient Systems*, *3*(1).

Kotsopoulos, D., Floyd, L., Khan, S., Namukasa, I. K., Somanath, S., Weber, J., & Yiu, C. (2017). A pedagogical framework for computational thinking. *Digital Experiences in Mathematics Education*, *3*(2), 154-171.

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*, *72*, 558–569. https://doi.org/10.1016/j.chb.2017.01.005

Kortuem, G., Bandara, A. K., Smith, N., Richards, M., & Petre, M. (2013). Educating the internet-of-things generation. *Computer*, *46*(2), 53–61.

https://doi.org/10.1109/MC.2012.390

Krajcik, J., Blumenfeld, P., Marx, R., & Soloway, E. (1998). Instructional, curricular, and technological supports for inquiry in science classrooms.

Krajcik, J. S., & Blumenfeld, P. C. (2006). Project-based learning (pp. 317-34).

Langtangen, H. P. (2006). *Python scripting for computational science* (Vol. 3). Berlin, Heidelberg and New York: Springer.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., … Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32. https://doi.org/10.1145/1929887.1929902

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin*, *41*(1), 260-264.

Machluf, Y., Gelbart, H., Ben-Dor, S., & Yarden, A. (2017). Making authentic science accessible—the benefits and challenges of integrating bioinformatics into a high-school science curriculum. *Briefings in Bioinformatics*, 18(1), 145–159. https://doi.org/10.1093/bib/bbv113

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational Thinking in K-9 Education. *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference - ITiCSE-WGR '14*, (January 2016), 1–29.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch

programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 16.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). *Programming by choice: urban youth learning programming with scratch* (Vol. 40, No. 1, pp. 367-371). ACM.

Harvey, B., & Mönig, J. (2010). Bringing "no ceiling" to scratch: Can one language serve kids and computer scientists. *Proc. Constructionism*, 1-10.

Mayer, R.E. (1989). The psychology of how novices learn computer programming. In E. Soloway & J.C. Spohrer (Eds.), Studying the novice programmer (pp. 129–159).

Mikhak, B., Lyon, C., Gorton, T., Gershenfeld, N., McEnnis, C., & Taylor, J. (2002, December). Fab Lab: an alternate model of ICT for development. In *2nd international conference on open collaborative design for sustainable innovation*.

Miles, M. B., Huberman, A. M., & Saldana, J. (2014). Qualitative data analysis. Sage.

McNeill, K. L., & Berland, L. (2017). What is (or should be) scientific evidence use in k-12 classrooms? What is (or should be) scientific evidence use. *Journal of Research in Science Teaching*, 54(5), 672–689. https://doi.org/10.1002/tea.21381.

McNeill, K. L., Berland, L. K., & Pelletier, P. (2017). Constructing explanation. *Helping students make sense of the world using next generation science and engineering practices (229-257). Arlington, VA: National Science Teachers Association. Berland, LK & Reiser, BJ (2009). Making sense of argumentation and explanation*.

McNeill, K. L., & Krajcik, J. (2008). Scientific explanations: Characterizing and evaluating the effects of teachers' instructional practices on student learning. *Journal of Research in Science Teaching: The Official Journal of the National Association*

for Research in Science Teaching, *45*(1), 53-78.

Martin, L. (2015). The promise of the maker movement for education. *Journal of Pre-College Engineering Education Research (J-PEER)*, *5*(1), 4.

Marder, M., Hughes, K. (2017, December 4). What Universities Can Do to Prepare More Computer Science Teachers. Retrieved September 10, 2018, from https://medium.com/@uteachinstitute/what-universities-can-do-to-prepare-more-computer-science-teachers-beed32d0559d.

Mishra, P., & Koehler, M. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record*, 108(6), 1017–1054.

NGSS Lead States, (2013). Next generation science standards: For states, by states.

National Research Council. (2012). *A framework for K-12 science education: Practices, crosscutting concepts, and core ideas*. National Academies Press.

Office of Science and Technology Policy. (2016). Preparing for the future of artificial intelligence [PDF document]. Retrived from https://obamawhitehouse.archives.gov/sites/default/files/whitehouse_files/microsites/ostp/NSTC/preparing_for_the_future_of_ai.pdf

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism*, *36*(2), 1-11.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.

Papert, S. (1993). The children's machine. *Technology review*. Manchester, NH, 96, 28-28.

Piaget, J. (1950). Explanation in sociology. *Sociological studies*, 30-96.

Przybylla, M., & Romeike, R. (2014). Physical Computing and Its Scope: Towards a

Constructionist Computer Science Curriculum with Physical Computing. *Informatics in Education*, *13*(2), 241-254.

Quintana, C., Reiser, B. J., Davis, E. A., Krajcik, J., Fretz, E., Duncan, R. G., & Soloway, E. (2004). A scaffolding design framework for software to support science inquiry. *The journal of the learning sciences*, *13*(3), 337-386.

Resnick, M., Berg, R., & Eisenberg, M. (2000). Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. *The Journal of the Learning Sciences*, *9*(1), 7-30.

Repenning, A., & Ioannidou, A. (2008). Broadening participation through scalable game design. *ACM SIGCSE Bulletin*, *40*(1), 305. https://doi.org/10.1145/1352322.1352242

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education - SIGCSE '10*, 265. https://doi.org/10.1145/1734263.1734357

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67.

Rode, J. A., Booker, J., Marshall, A., Weibert, A., Aal, K., von Rekowski, T., … Schleeter, A. (2015). From computational thinking to computational making. *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers - UbiComp '15*, 401–402.

https://doi.org/10.1145/2800835.2800926

Schwab, J. J. (1958). The teaching of science as inquiry. *Bulletin of the Atomic Scientists*, *14*(9), 374-379.

Selby, C., & Woollard, J. (2013). *Computational thinking: the developing definition*.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351-380.

Shein, E. (2015). Python for beginners. *Communications of the ACM*, *58*(3), 19-21.

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014a). Exploring the science framework and NGSS: Computational thinking in the science classroom. *Science Scope*, *38*(3), 10.

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014b). Computational thinking in high school science classrooms. *The Science Teacher*, *81*(5), 53.

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational researcher*, 15(2), 4–14.

Shulman, L. S., & Keislar, E. R. (Eds.). (1966). *Learning by discovery: A critical appraisal* (Vol. 5). McNally.

Stager, G. S. (2013, June). Papert's prison fab lab: implications for the maker movement and education design. In *Proceedings of the 12th International Conference on Interaction Design and Children* (pp. 487-490). ACM.

Stake, R. E. (2013). *Multiple case study analysis*. Guilford Press.

Smith, D. C., Cypher, A., & Tesler, L. (2000). Novice programming comes of age. Communications of the ACM, 43(3), 75–81.

Song, Y. (2014). "Bring Your Own Device (BYOD)" for seamless science inquiry in a primary school. *Computers & Education*, *74*, 50-60.

Tabet, N., Gedawy, H., Alshikhabobakr, H., & Razak, S. (2016, July). From alice to python. Introducing text-based programming in middle schools. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 124-129). ACM.

Tanenbaum, J. G., Williams, A. M., Desjardins, A., & Tanenbaum, K. (2013). Democratizing technology. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*, (September 2014), 2603. https://doi.org/10.1145/2470654.2481360.

Trower, J., & Gray, J. (2015, February). Blockly language creation and applications: Visual programming for media computation and bluetooth robotics control. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 5-5). ACM.

Vygotsky, L. S. (1978). Mind in society: The development of higher mental process.

White House. (2014, June 17). Presidential proclamation—National day of making. Retrieved from http://www.whitehouse.gov/the-press-office/2014/06/17/presidential-proclamation-national-day-making-2014

Wilcox, J., Kruse, J. W., & Clough, M. P. (2015). Teaching science through inquiry. *The Science Teacher*, *82*(6), 62.

Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering computational literacy in science classrooms. *Communications of the ACM*, *57*(8), 24-28.

Tisue, S., & Wilensky, U. (2004, May). Netlogo: A simple environment for modeling

complexity. In *International conference on complex systems* (Vol. 21, pp. 16-21).

Tsukamoto, H., Takemura, Y., Nagumo, H., Ikeda, I., Monden, A., & Matsumoto, K. I. (2015, October). Programming education for primary school children using a textual programming language. In *Frontiers in Education Conference (FIE), 2015 IEEE* (pp. 1-7). IEEE.

Wilkerson, M. H., & Fenwick, M. (2017). Using mathematics and computational thinking. *Helping students make sense of the world using next generation science and engineering practices*, 181-204.

Windschitl, M. (2000). Supporting the development of science inquiry skills with special classes of software. *Educational technology research and development*, *48*(2), 81-95.

Wing, J. (2011). Research notebook: Computational thinking—What and why? *The Link Magazine*, Spring.

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

Wing, J. M. (2006). Computational Thinking. *CACM*, *49*(3), 33–35.

Wolber, D. (2011, March). App inventor and real-world motivation. In Proceedings of the 42nd ACM technical symposium on Computer science education (pp. 601-606). ACM.

Yaşar, O. (2017). The essence of computational thinking. *Computing in Science & Engineering*, *19*(4), 74-82.

Yin, R. K. (2013). *Case study research: Design and methods*. Sage publications.

Appendix A: Interview and Observation Protocols

Teacher semi-structured pre-interview questions

Goals of the unit, Problems envisioned

1. What are your goals for this Smart Greenhouse unit?
   a) What are you goals of teaching science content?
   b) What are you goals of teaching coding or computation?

2. Did you teach any content relevant to this unit before?

   a) If yes, what are the content areas?
   b) How is this unit different from your previous curriculum?
3. Which parts of the unit do you think will work well?

4. What problems do you envision your students might encounter?
   a) What problems students may meet when they learn how to code the sensors?
   b) What problems students may meet when they work on their own designs of the greenhouse?
   c) How do you plan to help students if they struggle? What kind of support may be helpful for your implementation of the project?

Views about integrating coding with science learning and teacher support

5. Have you done any projects before that integrate coding with science learning?
   a) If yes, could you tell me more about them?
   b) In general, what do you think are the challenges to teach such projects in classrooms?

6. Other than the preparation sessions held by us, have you done extra work to prepare for the implementation?
   a) Did you play with the sensors and the coding on weekends or during your preparation periods ("preps")? Can you tell me more about it?

7. Any suggestions on how we can improve the preparation sessions for teachers who are interested in running the Smart Greenhouse unit?
   a) Did you have any problems during the preparation sessions? What are those?
   b) How can we improve the preparation sessions?

Views of computational thinking

8. Have you heard the term "computational thinking"? What do you think it means?

Teacher semi-structured post-interview questions

1. Could you tell me about how your impression of this project: What worked? What didn't work?
   a) Do you still think the idea of integrating science and coding is a good one? Why or why not?
   b) Do you think the project meet the goals of teaching science content? Why or why not?
   c) Do you think the project meet the goals of teaching coding? Why or why not?

2. What do you think the students learned?

3. What do you think was the biggest difficulty about trying to teach science and coding together? Did one supersede the other?
   a) If you did this again, what would you do differently?

4. We are interested in learning about your views about the support we provided. What types of support were helpful? What were not helpful? Why were they helpful/not helpful? (Please specifically probe these points)
   a) What's your opinion about Python? How was your experience learning it? How was your experience teaching it?
   b) Was the curriculum that we provided helpful/unhelpful for you to prepare your lessons? Why?

5. We were very lucky to have you two great teachers and you did the final project slightly differently. Could you tell me why you did what you did? Why did you make these instructional decisions?

6. To what extent do you think the PD sessions were effective for the classroom implementation of the program?
   a) How could we improve these PD sessions for teachers who will run the project the first time? (Probe: anything we should include in the PD? Anything that is unhelpful?)
   b) For the PD this summer, what do you hope to achieve?

7. What were the coding concepts that you learned? How did you teach them? Would you teach these concepts differently next time? Do you see any connection between these concepts and science/other disciplines?

8. Think about the teaching you did with coding and your teaching of science, are these teaching experiences similar or different? What are the similarities and differences?

Student semi-structured pre-interview questions

(Approximately 5~8 mins):

Views about coding, interest, and prior coding experience:

1. What comes to mind when you hear the word "coding"?
2. Have you done coding before? Can you tell me more about it? (make sure that students talk about their prior coding experience: when, where, what, and why)
3. Are you interested in coding? Why or why not?
4. What do you think coding can do?
    a. What do you want to do with coding?

Views about people who do coding:

5. What kind of jobs do you think involve coding? Can you provide a couple of examples?
6. Do you think scientists do coding in work? Why or why not?

Student semi-structured post-interview questions

The purpose of the interview is to get to know more about your experience of this project, so that we can improve the project for next year's run. Your answers to the questions will have nothing to do with your final grades. If you don't feel comfortable with any of the questions or want to stop the interview at any time, please let me know.

******Please finish at least Questions 1 to 6*****

1. What do you think you have learned from the smart greenhouse project?
2. [Make sure that the interviewee has his/her greenhouse on the table]
   Could you tell me how your greenhouse works?
    a. Which parts of your greenhouse are you most proud of? Why?
    b. (After the students have described their greenhouse, identify a sensor (probably a temperature/humidity sensor or a light sensor) in the greenhouse, and ask)
       So you told me that you used xx sensor.
        i. Why did you use the sensor? Why is that important for your (basil/lettuce/cilantro)?
        ii. Can you describe the code that you wrote to do this? (in coding language?)
3. Can you tell me about a problem that you ran into while you were coding? How did you fix that problem?
    a. (Did you ever use error messages to help you fix your code?) How helpful/unhelpful were these error messages for you?
4. Now you have completed the project, do you see yourself wanting to learn more about coding? Why?
5. What else do you think you can do using the science and coding you have learned?
    a. Now that you learned how to program the MCU board and the sensors, what other applications can you think of with them?
6. When you heard that you were going to do a coding project, what did you think? Has this perception changed? Why or Why not?
    a. (If the student says yes) Which part of this smart greenhouse project changed your perception?
7. Do you want to take your greenhouse home? Why? What else would you like to do with it?
    a. (Optional Prompt) If you had more equipment, what else would you like to automate in your greenhouse?
8. Do you think scientists do coding in work? Why or why not?
    a. What role does coding play in science?
    b. Did this project change your views on how scientists use coding?

Did you tell anyone at home about the project? IF yes, who? How did you describe this to them?

Classroom Observation Protocol

Teacher_____ Period_____ Date_____ Observer_____

Student Names_____

Time	What did students do? What did teachers do?

Summary

Student engagement: What did students work on?	
What challenges or excitement did students experience?	

Appendix B: Comparison of Hardware Platforms

	Raspberry Pi	**Arduino**	**ESP8266**	**Micro:Bit**
Architecture	Micro-computer		Microcontroller	
Cost	$35.00	$22.00	$4-$15	$14.95
Languages	Python, C/C++*	Arduino	MicroPython, Arduino, Lua	MicroPython, JavaScript
Connectivity	WiFi, Bluetooth, Ethernet	No**	WiFi	Bluetooth***
Interactive Programming	Yes	No	Yes (w/ MicroPython)	Yes (w/ MicroPython)
Support for block-based programming	Yes	Yes	Yes***	Yes***
Community Support	Excellent	Excellent	Great	Great
Scalability and Sustainability	Excellent	Excellent	Great	Good
Open Source	Yes	Yes	Yes	Yes
Grove Compatibility	Through external boards	Through external boards	Supported by the Wio Link board	Through external boards
Chromebook compatibility	N/A	Yes	No	Yes
IDE** **Support**	Excellent	Excellent	Excellent/Limited when using MicroPython	Excellent

Note:
*: Technically most languages compatible with Linux run on Raspberry Pi. Only the most popular languages are listed here.
**: Can be added through extension boards for additional cost.
***: Not available when using MicroPython.
****: Integrated Development Environment (IDE): software programs that provides functionalities for programming, such as syntax highlighting, testing, auto-complete.

Appendix C: Structure of the Curriculum

Goal: Design a smart greenhouse with a microcontroller and sensors to collect data and answer a relevant research question
Curriculum website: http://growthings.netlify.com/unit1/lesson1/

Day	Topic	Goals	Learning Activities	CS/Tech	Reading Materials
1	Course Overview	1. Learning the science of smart, automated greenhouses 2. Understanding microcontrollers, Python, and import statements	• Introduction to real-world greenhouses • Interactive coding in Python	Using Python to perform basic arithmetic operations on the microcontroller	How do greenhouses work
2	Light and Plant Growth	1. Understand how colors of light affects plant growth 2. Program the LED strip to mimic professional grow lights	• Watching a video on photosynthesis, Understanding the light spectrum, Programming the LED strip	Create instances of objects, use functions (methods) to modify states of objects, and passing function arguments (Parameterization), using lists (data structures) to control colors of light	Can Plants "see" light?
3	Light and Plant Growth	1. Understand how intensity of light affects plant growth and the relationship between light intensity and distance to light source/color 2. Program the digital light sensor to measure light intensity in lux	• Use the light sensor to measure light intensity from the LED strip. • Hypothesize the nature of the relationship between distance to the light source and light intensity. • Graph the light sensor readings at different distances	Continue using functions (methods) with return values to read light intensity values in lux. Use loops to measure data at specific intervals	Light intensity and duration impact on plant growth

			to test the hypothesis and understand the relationship		
4	Temperature/Humidity and plant growth	1. Understand how change of temperature affects (relative) humidity (think as a system) 2. Program temperature and humidity sensors to measure these two values	• Watching a video on how temperature/humidity affect the photosynthesis/respiration • Learn to collect data and automatically transmit data to other devices	Understand how data is communicated between networked devices	Temperature/humidity and plant growth
5	Treasure hunt with live data visualizations	Making scientific hypotheses and using scientific reasoning with empirical data	• Game: Treasure hunt. select groups of students hide temperature/humidity sensors somewhere in the classroom for others to locate them using live data visualizations	Read real-time data visualization and make scientific arguments	N/A
6	Manipulation of temperature/humidity in greenhouses	1. Design an experiment to test hypotheses 2. Use actuators (servos, relays, and fans) if statements to achieve automation	• Come up with a scientific hypothesis on how variables (temperature, humidity, light intensity, etc.) affect plant growth. Design a smart greenhouse that can manipulate these variables and collect data that helps support the hypothesis	Use if conditionals to achieve automation using sensor data and actuators	N/A
7	Design and build the greenhouse	Students continue to design and build smart greenhouse that could help them support their hypotheses or answer research questions that they came up with in Lesson 6.			
8					
9					

10	Gallery Walk	Students present their design of greenhouses and answer questions from their teachers/peers about their greenhouses	Students also explain the programming skills and concepts that they developed during the course	N/A