Pepperdine University

Graduate School of Education and Psychology

BROADENING PARTICIPATION IN COMPUTER SCIENCE: THE LIVED EXPERIENCES AND PERCEPTIONS OF K-12 TEACHERS WHO RECONTEXTUALIZE CODE OUTSIDE COMPUTER SCIENCE

A dissertation submitted in partial satisfaction

of the requirements for the degree of

Doctor of Education in Learning Technologies

by

Joseph Eli Chipps

January, 2020

Linda Polin, Ph.D. – Dissertation Chairperson

This dissertation, written by

Joseph Eli Chipps

under the guidance of a Faculty Committee and approved by its members, has been submitted to and accepted by the Graduate Faculty in partial fulfillment of the requirements for the degree of

DOCTOR OF EDUCATION

Doctoral Committee:

Linda Polin, Ph.D., Chairperson

Judith Fusco Kledzik, Ph.D.

Brian Foley, Ph.D.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

DEDICATION

For my wife, Jeannie, the first recontextualizer in my life.

Thank you for the love and support throughout this process.

ACKNOWLEDGEMENTS

I would like to thank Dr. Linda Polin, the chair of this dissertation, for her enthusiasm for learning theory, mentorship during the dissertation process, and friendship. Thank you Dr. Brian Foley and Dr. Judi Fusco for your advice, feedback, and support. You were a wonderful committee that continued to push me, and I am grateful for you.

Thank you, Dr. Jane Margolis, Dr. Joanna Goode, and Gail Chapman for bringing me into the world of computer science education and opening my heart to social injustice. Additionally, I would like to thank Dr. Deborah Fields and Dr. Yasmin Kafai for continuing my identity shift toward researcher by treating me as a colleague.

I have to thank my mother, Judith Chipps, without whom none of this would be possible. Thank you, Jessie Chipps, for being the best sister ever. I am also grateful for my father, Stephen Chipps.

A big thank you to all of the members of Cadre 22 for sharing the ride. In addition, I would like to thank Dr. Mark Chen, Dr. Cameron Sublett, and Dr. Kay Davis for their high-quality instruction.

Lastly, I am grateful for the Earl V. Pullias Endowed Scholarship.

VITA

EDUCATION

2020	Pepperdine University, Malibu, California Doctor of Education in Learning Technology
2013	California State University, Northridge, California M.A. in Education with a Focus on Computers and Technology in Secondary Education
2010	California State University, Northridge, California Single Subject Teaching Credential - Mathematics
2008	Vassar College, Poughkeepsie, New York B.A. in Mathematics, Music

TEACHING EXPERIENCE

- California State University, Northridge SED 514: Computers in the Instructional program — 2014-present
  - Instructed new teachers in the credential program on technology use in the classroom.
- California State University, Northridge SED 614: Computers in Mathematics and Science Teaching — 2016-present
  - Instructed teachers in the Master's in Mathematics Education program on research-based technology use
- Granada Hills Charter High School; taught courses in mathematics and computer science — 2008-present

OTHER COMPUTER SCIENCE EDUCATION EXPERIENCE

- Computer Science Standards Advisory Committee — 2017- 2018
  - Collaborated to develop the California Computer Science State Standards

- PD Facilitator for Exploring Computer Science 2014-present
  - Trained by Jane Margolis, Gail Chapman, and Joanna Goode to facilitate ECS PD.
  - Facilitated PD for ECS teachers in California, Nevada, and South Carolina
- Computer Science Standards Focus Group 2016
  - Served as a focus group member at the San Diego County Office of Education
- 100 CS Teacher Workshop
  - Participated in an NSF ECS/CSP workshop in Washington D.C., and was invited to the White House during CS education week 2014

PUBLICATIONS/PRESENTATIONS

- Foley, B., Chipps, J. (2019, October 3). Integrating computer science into science and math instruction. Round table facilitated at Cyberlearning 2019: Exploring Contradictions in Achieving Equitable Futures, Alexandria, VA.
- Chipps, J, Fraizer, L., (2017, September 22). Hack the gatekeepers: Increasing participation in computer science. Working paper presented at the International Organization of Social Sciences and Behavioral Research, Atlantic City, NJ.
- Nobles, M., Fraizer, L., Chipps, J. (2017, September 22). Moving beyond the curriculum: Awakening purpose. Working paper presented at the International Organization of Social Sciences and Behavioral Research, Atlantic City, NJ.
- Chipps, J. (2016, September 24). E-Textiles as an equitable substitute for robotics. Presentation at Computer Science Teachers Association, Inland Empire, CA.
- Chipps, J. (2016, September 16). Showcasing e-textile products and procedures. Presentation at E-Textiles Fair at UCLA, Los Angeles, CA.

LEADERSHIP EXPERIENCE AT GRANADA HILLS CHARTER HIGH SCHOOL

- Math and Computer Science Department Chair 2014-present
  - Developed four-year computer science pathway, focusing on equity and access
  - Managed department-wide creation of course sequencing and unit plans under Common Core
  - Organized professional development for teachers to engage in 21st century learning models

- STEM Coordinator 2012-2015
  - Created four-year program culminating with an independent STEM research project
  - Organized the creation of cross-curricular, intra-departmental projects
  - Brought the UCLA-X course, Exploring Computer Science, to all STEM freshmen
- Summer Transition Academy Math Coordinator 2010-2016
  - Developed 4-week and 2-week curriculum for incoming freshmen
  - Gathered data on incoming freshmen, and used data to provide academic and social support
  - Trained and led all teachers in program

RESEARCH INTERESTS

- Educational Technology, Technology and Communities, Virtual and Augmented Reality, Increasing Participation in Computer Science Education, Computational Thinking and Practices, Accessibility

- Dissertation focus: My dissertation serves to describe the experiences and perceptions of K-12 teachers who intentionally re-contextualize programming into non-computational spaces.

ABSTRACT

Underrepresentation in computer science education is compounded by a K-12 system that is unable to make significant changes to build a sustainable pipeline, a shortage of qualified teachers, a masculine culture of computation that creates barriers for women and people of color, and a lack of opportunities for teachers to learn computational thinking. Rather than increasing participation in computer science courses, which is not broadly scalable given the limitations of the K-12 system, some researchers have focused on integrating computing and computational thinking into pre-existing, non-computational courses, or recontextualizing programming. The purpose of this exploratory phenomenological study was to understand the perspectives and lived experiences of K-12 educators who have intentionally integrated programming into another K-12 field. Data were collected from eight informants using semi-structured interviews and supporting materials from their classrooms. All of the informants reported that they used coding as a tool for enhancing students' experiences and learning.

The study found that there are two types of recontextualizers, *Communicators* and *Weavers*. Communicators embed coding within projects where students communicate their learning through personalized artifacts, and a teacher in any subject matter can recontextualize programming via Communicator methods. Weavers integrate coding into their content standards, and are currently typically math and science teachers. Communicators and Weavers work within different activity systems that mediate their practice. The results of this study have significant implications for colleges of education, computer science education, educational technology, and teachers' systems of activity.

Keywords: computer science education, teacher pedagogy, recontextualizing programming, communicators, weavers, underrepresentation, K-12 education, computational thinking.

Chapter One: Introduction

Computer science education has stabilized at a critical juncture. At both the high school and college levels, enrollment in computer science courses is relatively weak (Archer et al., 2016), and skewed toward a white, male, student population (Margolis, Estrella, Goode, Holme, & Nao, 2008). Despite a large-scale, multifaceted approach to engaging underrepresented students in computer science, the problem persists (Fields, Kafai, & Giang, 2016; Fields, Lui, & Kafai, 2017a; Fields, Vasudevan, & Kafai, 2015; Goode, Flapan, & Margolis, 2018; Goode, Margolis, & Chapman, 2014; Kafai, Fields, & Searle, 2014a; Margolis, Goode, & Chapman, 2015; Perković, Settle, Hwang, & Jones, 2010; Rennert-May et al., 2012; Resnick et al., 2009; Settle, Goldberg, & Barr, 2013). There appear to be four critical, systemic, barriers to increasing the number and diversity of students engaging with programming and computational thinking: culture; curricula; the number and qualifications of teachers; and a self-reinforcing K-12 educational system.

These barriers cannot be considered in isolation; rather, they are components of larger systems. Interacting factors within schools such as counselor-student communications, graduation requirements, curriculum adoption, and teacher training opportunities influence the availability and attitudes toward computer science courses. Furthermore, there are district and state level mandates that influence where funds are appropriated, which courses are significant with respect to future school funding, and which subjects deserve a specialized credential that serve to mediate how teachers and students approach computer science. Finally, coding textbooks and programming languages and environments are developed by industry veterans, and mediate the teaching and learning of computer science. Consequently, computer science education is situated in multiple contexts, and focusing on one component of the system requires mention and analysis of the others.

Culture

The culture of practice within STEM fields excludes women and, in particular, women of color (Ong, Wright, Espinosa, & Orfield, 2011), but is particularly exclusive in computer science, which has been called a locked clubhouse (Fisher & Margolis, 2002; Margolis et al., 2008). The practice and norms of the occupation of computing has been referred to as masculine; that is, there exist different expectations of men and women that inequitably impact different populations (Faulkner, 2001; Frehill, 2004; Wacjman, 1991; Wajcman, 2009a; Woodfield, 2002). Furthermore, the occupation of computing creates a chilly climate for marginalized populations (Mellström, 1995; Rasmussen & Håpnes, 1991), which has often created barriers for those not aligned with masculine technological culture.

The occupational culture of computing transferred to K-12 learning spaces through the cultural artifacts that define the practice of learning. The symbols and tools of computer science education such as curricula, programming languages and environments, belief systems of faculty and students, and the spaces we create for learning are mediated by a culture defined by the gendering of computer programming in the 1970's (Ensmenger, 2010; Fitzsimons, 2002). When organizations structure division of labor, norms and practices become gendered (Martin, 2003; Schilt, 2006; West & Zimmerman, 1987), which negatively impacts women and subdominant groups in the organization.

Curriculum

A key part of recent computer science education has been the development of new curricula in K-12 that aim to support and engage underrepresented minorities. K-12 courses like

Exploring Computer Science, *AP Computer Science Principles*, and *The Beauty and Joy of Computing* created activities that are intended to be accessible to students without much background in computers, and are intended to be engaging and relevant to students' lives outside of school. These courses have gained popularity to the point where in many high schools, there is more demand for the course than there are teachers prepared to teach them (Cross, 2017). These courses have dramatically increased the number and diversity of students who are taking CS courses (Astrachan, Gray, Beth, Osborne, & Lee, 2014; Cuny, 2015; Goode et al., 2018).

Despite recent gains in underrepresented minorities taking introductory computer science courses, the numbers of students taking the most advanced course, AP Computer Science A, are still minimal. Only 0.05% of the approximately 1.9 million California public high school students took the AP CS A exam in 2017. Students of color comprise over 60% of California's high school-aged population (53% Latinx, 6% African American, ~1% American Indians/Alaska Natives). However, just 15% of AP CS A test-takers were Latinx, 1% African American, and only 5 American Indians/Alaska Natives passed the test out of the 10,268 test-takers from California in 2018 (AP Program Participation and Performance Data 2018, 2018). Low participation rates among students of color have resulted in computer science not being offered at 75% of schools nationwide with the highest percentage of underrepresented students of color and only 2% of schools with large ratios of students of color offering AP Computer Science A (Margolis & Goode, 2016).

Numbers and Qualifications of Teachers

Not only are few schools offering computer science, but also a remaining major obstacle for CS education is the lack of a CS teacher pipeline (Cross, 2017). Although the state of California recently developed the first set of K-12 standards for computer science (California

State Board of Education, 2018), there is currently no credential for teaching computer science. In addition, only 22 states have adopted K-12 computer science standards, and only 13 states have state-approved preservice preparation programs for computer science teachers in higher education institutions, and California is not one of them (AP Program Participation and Performance Data 2018, 2018). In fact, training programs such as credentialing programs within schools of education currently do not offer courses in computer science-specific pedagogy, as currently exists for math and science education; rather, for most states, including California, the role of training computer science teachers falls on private organizations and grant-funded projects. State and district initiatives to increase computer science education have historically relied on partnering with organizations with pre-designed curriculum and a professional development model for training teachers (Margolis et al., 2015). Still, only 19 states have provided funding for computer science professional development (AP Program Participation and Performance Data 2018, 2018).

The teachers chosen by schools and districts to teach computer science typically do not come from the computer science field; they are credentialed in another subject such as math, science, art, social studies, or career technical education (Goode, 2007a). This results in computer science teachers with minimal content knowledge relying on the curriculum and tools developed by those in the industry to engage students. In addition, students are tracked into computer science courses inequitably via the unequal course offerings and expectations among teachers, schools, districts, and states (Goode, Estrella, & Margolis, 2006). A system of inequity has created an educational field with a lack of teachers, inequitable pathways and opportunities for students, and belief systems that block access for many students. Teachers who use programming and computational thinking in their classes can either continue systems of bias and repression or become agents of change within their communities to increase diversity. To become agents of change, teachers must be given appropriate tools and training (Goode, 2007a).

The challenge of diversifying K-12 computer science education is further impacted by the lack of a computer science credential and teacher training opportunities. Teachers fortunate enough to participate in an NSF-funded project designed to train teachers and increase underrepresentation are given tools to address diversity in the computer science classroom (Astrachan et al., 2014; Goode, Chapman, & Margolis, 2012); however, these opportunities are not wide-spread to all parts of the country. Due to a lack of computer science credentials across states, colleges of education have no legal requirement to train computer science teachers in content-specific pedagogy and challenges. As a result, the only computer science training opportunities are provided by projects funded by grants, which are not sustainable.

Self-Reinforcing K-12 System

Margolis et al. (2008) paint a picture of a K-12 educational system not flexible enough to handle broad computer science curricula due to a lack of teachers, technology, technological support, deficit systemic belief systems regarding race and gender, and state and federal mandated legislation that places increased attention in other areas of education. Despite the need for more well-trained professionals entering the computer science field and a generation of students who purport to want to pursue a career in computer science (Margolis et al., 2008), the current K-12 educational system is ill-prepared to provide adequate access, specifically for underrepresented youths. In order to increase access for all, computer science education must begin in Kindergarten at all schools and continue until college, but this broadening of participation can only be done sustainably (Wolz, Stone, Pulimood, & Pearson, 2010), and adding computer science classes in K-12 schedules in order to address another graduation requirement is not sustainable given the current K-12 educational system. Computer science opportunities currently exist within extracurricular spaces such as summer camps and after-school programs; however, access to those programs is privileged, and has traditionally favored affluent white boys (Goode et al., 2006; Simpson, 2001).

Unless programming becomes mainstream within formal K-12 educational systems, underrepresented groups will continue to experience lack of access to computer science education. Increasing participation in computing requires a sustainable and broad endeavor that accepts the premise that all students can engage in computational thinking and deserve to do so. Given the inability to add computer science into K-12 schedules due to the inflexibility of the K-12 system, this requires moving computer science away from a separate and segregated entity and finding applications for computing within non-computational fields. Furthermore, it requires thinking beyond just the curriculum and focusing on the cultures of classrooms and the communities that engage in practice in K-12 spaces (Pulimood & Wolz, 2008; Wolz & Pulimood, 2007).

There are K-12 teachers in non-computational fields who use computer programming in their classes. For example, a colleague uses simulations with block-based code to engage students in her chemistry classes. Another colleague who teaches Chinese has her students develop animations in Scratch, where characters have conversations with each other in Mandarin. A math teacher has her students develop functions in Python for each of the procedural concepts they learn. None of these teachers have a formal education in computer science, and yet they leverage the power of computer science to support students' construction of concepts in their respective classes. Understanding the experiences and perceptions of K-12 teachers who engage students in their non-computational classes using computer programming may provide insights into what it means to teach computer science, the spaces in which computer science education exists, and requirements for teacher preparation programs.

Expanding Participation in Computing

The activity of programming, as it occurs in school settings, is embedded within a particular and traditional cultural milieu that research suggests has an exclusionary impact on many students. However, increasingly, teachers in subject matters other than computer science or programming courses are including coding activities in their coursework. For example, programming an animated storyboard for a journalism project requires students to construct deep knowledge of arrangement of events and how that arrangement tells a story. This sort of pedagogical context for coding opens up opportunities for students to view and experience programming as a meaningful and valuable skill, embedded within a different cultural milieu than the culture of computing. While still relatively uncommon, the efforts of teachers in subject matter other than computer science offer insight into the ways in which programming can be integrated into meaningful curricular context. It is then worthwhile to look at the activity of computer programming within non-computational spaces in order to reevaluate what coding is and when people code.

This study focused on formal K-12 spaces where computer programming is intentionally integrated into non-computational fields in order to give new meaning to those engaged in the practice of programming. More specifically, the purpose of this exploratory qualitative study is to understand the perspectives and lived experiences of K-12 educators who have intentionally integrated programming into another K-12 field.

Research Question

How do K-12 teachers describe their experiences and perceptions of their efforts to redefine programming in a more engaging landscape such as a non-computer science class?

Significance

This study aimed to provide insights into how teachers develop the content and pedagogical knowledge required to integrate programming and computational thinking into their respective classrooms. There are three key populations for which this study holds significance: schools of education that want to support 21st century learning; K-12 teachers who are interested in using programming in their class but lack the tools and means of doing so; and future students who will begin to view computing as a part of 21st century skills rather than an elite subject.

Schools of education. Although teacher preparation programs continue to support 21st century thinking and learning, they do not yet focus on computational thinking and programming across the curriculum. This study could provide ways of imagining teacher preparation programs for the purpose of initiating teachers to the possibility of integrating programming in their classrooms as well as providing support and direction. Results from this study could also persuade departments of education to develop a secondary certification in computational thinking and programming, and could inform credential programs on best practices for supporting teachers who wish to integrate computing into their classroom.

K-12 teachers. Currently, K-12 teachers who wish to begin integrating computational thinking into their classrooms rely on the Next Generation Science Standards and Common Core State Standards, both of which mention the importance of computational thinking (Brown & Concannon, 2018; Porter, McMaken, Hwang, & Yang, 2011). Additionally, the new California Computer Science Standards (California State Board of Education, 2018) are designed to be implemented in general education classes for teachers who wish to incorporate computer science in their classrooms. Although standards support content development, they do not give teachers information on pedagogical knowledge, how to manage technology, how to design the classroom space, how to manage students, or where to start looking for ideas. By describing the lived experiences of K-12 teachers who have integrated programming in their classrooms, this study aimed to provide a framework for other teachers who wish to do the same.

Students. This study also positioned computer science as an invaluable skill that can be used within any other field. By positioning computer science as a tool for practice rather than a separate and segregated field, students in non-computational classes can engage in computing and computational thinking, contextualized on the practices and content of the respective field (Gadanidis, 2015; Papert, 1980; Weintrop et al., 2016). This has immense implications for providing access for all to computer science, and changes how computer science is considered in the literature.

Summary

Contextualizing programming has been shown to increase participation among women and students of color. To integrate programming into another non-computational field would contextualize programming beyond the imaginings of contextualization. This study therefore defined recontextualization as the purposeful integration of programming into a non-computational field. The purpose of this study was to describe the experiences and perceptions of K-12 teachers who recontextualize programming.

Chapter Two: Literature Review

Underrepresentation in computer science not only exists at the industry level, but also in the K-12 and collegiate educational spaces. Scholars initially used programming in the K-12 educational space as a means of supporting the acquisition of mathematical knowledge (Feurzeig, Papert, & Lawler, 1969; Papert, 1972; Solomon & Papert, 1976). Over time, and with the support of entities such as College Board, the object of programming changed from a tool for mathematics content acquisition to a subject and content area of its own. As programming shifted to a content area of its own, the tools and symbols used by the professional community transferred to the K-12 space, in turn mediating the learning and teaching of programming (Resnick et al., 2009). As a result, the masculine occupational culture of computing transferred to the K-12 space (Margolis et al., 2008).

Consequently, researchers have developed interventions at the K-12 level to increase underrepresented youths' participation in computer science education. Program languages designed for learners such as Scratch, Alice, and App Lab are visual programming language that offer the ability to perform complex operations with ease (Tsukamoto et al., 2016). New curriculum funded by the National Science Foundation like Exploring Computer Science and AP Computer Science Principles were intentionally designed to increase participation by rejecting the traditional programming-centric structure of computer science courses in favor of contextualizing computing concepts on a broad-range of topics (Cuny & Jan, 2015; Goode et al., 2012). Despite advancement in curriculum and the tools used to teach programming, the current K-12 educational system is not prepared to accommodate a national and wide-spread endeavor to mainstream computer science in schools as there are too few computer science teachers and computer science is not a core subject (Cooper, Pérez, & Rainey, 2010; Margolis et al., 2008).

Recontextualizing programming, or integrating programming and computational thinking into non-computer science fields, has great potential, as it would remove the need for a systemic overhaul that would require changes in public policy. Contextualizing programming follows the design theory of courses like Exploring Computer Science and AP Computer Science Principles in that programming is contextualized within another domain. Additionally, contextualizing programming aligns with new national and state standards such as the Next Generation Science Standards (Brown & Concannon, 2018) and Common Core State Standards (Porter et al., 2011) as they specifically reference computational thinking as part of the practice of the subject. Furthermore, the recently written California State Computer Science Standards are designed for stand-alone CS classes as well as traditional K-12 courses that integrate computer science into the curriculum (California State Board of Education, 2018).

Despite some studies focused on schools and organizations that recontextualize code into non-computer science fields (Hug, Grunwald, Lewis, Goldberg, & Feld, 2012; Perković et al., 2010; Rennert-May et al., 2012; Settle et al., 2013), there is little literature regarding the experiences and perceptions of teachers as they make the change in order to understand their needs. Since the activity of computing is a complex system that is practiced within a large community, in order to understand the needs of teachers who recontextualize programming, a method of understanding the system in its entirety is warranted.

Activity theory provides conceptual tools to understand multiple perspectives and networks of activity systems (Engeström, 2011), and is a useful lens for describing the practice of programming within larger communities. Activity theory is a lens for understanding the relationship and mediational effect of activities, their cultural surrounding, and the broader culture in which they exist (Bødker, 1997). As there are specific sociocultural tensions experienced by those engaged in programming, it is worth analyzing the current and historical trajectory of the elements within the activity system to understand the source of the tensions. Specifically, activity theory will be used to illustrate how the tools and symbols that mediate teachers' activity while teaching code are embedded in the occupational culture of computing. Activity theory expands the actor–action relationship to include factors such as the environment, history, and, more pertinent to this study, cultural tools such as curricula and programming environments, that the individual uses (Kuuti, 1996). An activity consists of interacting components and the relationships between them. The components of an activity include the subject, object, mediating artifacts, division of labor, rules, community, and outcomes. The connections between components is often constructed as a triangle, where lines indicate interactions between the elements of the activity system (Engeström, 2000), shown in Figure 1.

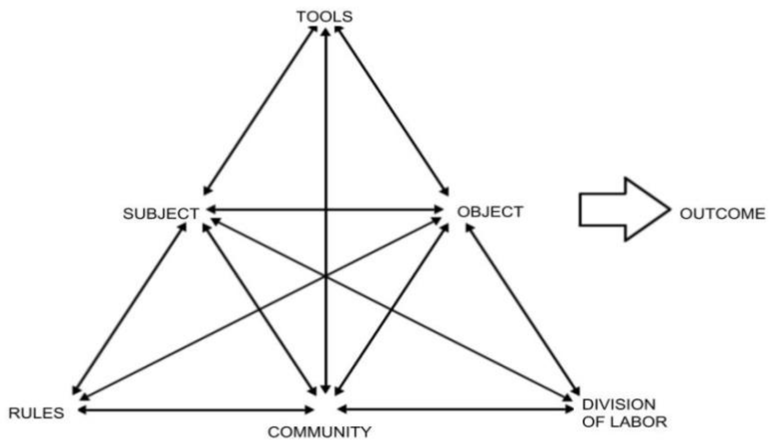


Figure 1. Activity theory triangle.

An activity is bounded by historically standardized rules that define appropriate behaviors. The rules and norms that govern activities are defined by a wider community that shares a common practice and ideas of what it means to participate in the practice (Engeström, 1987). The community is comprised of individuals or groups who share the same object, but exhibit different roles within the community with various functions, or a differentiated division of labor. In this study, the object of activity was increasing workforce eligible graduates in computer science for the purpose of broadening participation in computer science. By inspecting all actors in the community, activity theory provides a lens for understanding how participants with different perspectives and roles in the community can collaborate and develop new knowledge and tools, motivated by the object. The subject and object relationship is then mediated by the knowledge and tools constructed by the community as well as the relationship between the subject and the community (Engeström, 2001). Activities are the cultural-historical result of work and organizations, and are distributed through the roles and hierarchies defined by a division of labor.

The Cultural-Historical Activity of Programming in Industry

Underrepresentation in computer science has significant costs. Not only do women forego valuable opportunities, but also technology suffers design flaws due to a lack of diverse perspectives during development (Fisher & Margolis, 2002; Margolis et al., 2008). It is therefore of utmost importance to understand underrepresentation for the purpose of increasing diversity in the computer science workplace. Figure 2 shows the activity system of an industry worker leader who wishes to increase the computer science workforce, where red lines represent tensions between components.

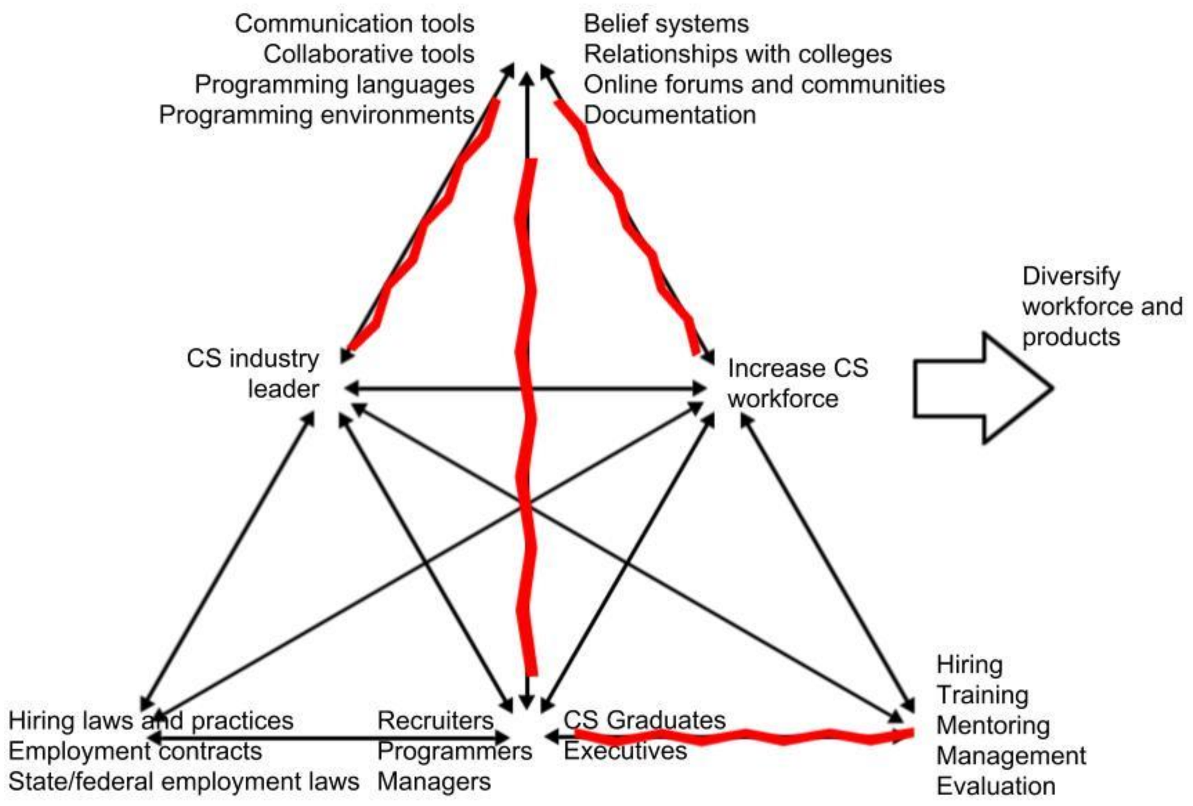


Figure 2. Activity system for CS industry leader to increase the CS workforce.

An activity system is a model for how components of the system mediate the subject's trajectory toward the object. The tools and knowledge of the community act upon the subject and community, and in turn mediates the subject's behavior and practice. Implementing activity-theoretic analysis to computer science provides the language to describe the relationship between technology and the community that develops the technology in context with the unit of analysis, which here is the activity itself. As such, the organization and technology are both elements of a system that mediate each other (Karanasios, Allen, & Finnegan, 2015). Human activity is always situated in context within a larger activity, and to inspect any element of the system, be it technology or subjects outside of that context is inadequate (Spasser, 1999).

Object. The object of an activity is defined as the thing that people are trying to transform, be it a project, focus, group, or problem situation (Blackler, 2009; Engeström et al., 1999). Objects are not static; rather, they are dynamic and active, constantly in motion and flux as they are socially constructed and evolving (Blackler, 2009). For this study, we considered the object to be increasing workforce eligible graduates in computer science.

One effective method of understanding an activity is to analyze the development of an object over time (Kaptelinin & Nardi, 2012). Data from workforce demographics show that women's underrepresentation in the information technology (IT) sector is extensive, multifaceted, and persistent. Women represented 25% of the U.S. IT workforce in 2015. Of those women, 3% are African American, 5% are Asian American, and 1% are Hispanic (Ashcraft, McLain, & Eger, 2016). Not only is there underrepresentation in the IT workforce, but there is also considerable underrepresentation at the senior level of IT. Women comprise 9% of IT leadership around the world (Ellis & Heneghan, 2016). Similarly, Ashcraft and Blithe (2010) report that women represent 9% of U.S. executive and senior management IT positions. Furthermore, at the executive IT level, women of color are extraordinarily rare.

Community. Schein (2010) defines culture as:

> a pattern of shared basic assumptions that was learned by a group as it solved its problems of external adaptation and internal integrations that has worked well enough to be considered valid and, therefore, to be taught to new members as the correct way to perceive, think, and feel in relation to those problems. (p. 18)

Culture allocates norms for how people behave and appear as a measurement of appropriateness, and consequently, defines those who are unsuitable for an organization. Culture also determines the fitness of individuals within organizational hierarchies (Deal & Kennedy, 2000; Schein, 2010). Newcomers to an organization or role learn their craft via improvised moments of practice, wherein the subject is enculturated in a process that enables the individual to accept a role that fits both organizational and individual needs. Veteran practitioners function as insiders who help newcomers become veterans themselves, which implies that they are effective organizational members (Lave & Wenger, 1991).

Occupational culture. Guzman and Stanton (2009) assert that occupational cultures "include the basic assumptions, cultural forms, ideologies and behaviors that grow uniquely in the context of a particular occupation" (p. 160). Occupational culture can span multiple organizations as well as transcend the broader organization. The occupational culture of computer science, then, continues to be produced by the activity of those engaged in it. We must first analyze the occupational culture of computer science activity in order to understand how it transcends industry and organizations, and is embedded within the activity of programming.

By Guzman and Stanton's (2009) definition, information technology (IT) occupational culture includes the assumptions, ideologies, and behaviors of those engaged in IT professions. Guzman, Stam, and Stanton (2008) describe IT occupational culture as following: pervasive use of techno-speak; the demand for extreme and unusually long hours; feelings of superiority compared to users; a lack of formal work rules; and an obsession with IT use, even in non-work spaces. Further studies portray IT culture as principally white and male, sexist, misogynistic, anti-social, highly competitive, and aggressive toward women (Bartol & Aspray, 2006; Hewlett Luce, & Servon, 2008; Todd, Mardis, & Wyatt, 2005; Wentling & Thomas, 2009).

Division of labor. In the 1960's-1970's, before anti-discrimination legislation, software was professionalized (Fitzsimons, 2002), which had damaging results for women. During the hiring process, managers used a variety of aptitude tests and personality profiles, which over time constructed the ideal computer programmer to be a highly intelligent, socially inept male (Ensmenger, 2001; 2010). Fitzsimons (2002) and Ensmenger (2010) argue that the hiring process during the early stages of the professionalization of programming initialized the masculine occupational culture of computing because it created a division of labor and organizational hierarchy that tended to limit women to low-level jobs while favoring the anti-social male.

Professionalization. Although professionalization increased the value of programming and the individuals who do it, the irrevocable damage it did for women in the form of entry barriers is part of the same narrative. Over time, computer programming became a predominantly and stereotypically male activity and increasingly inhospitable to women (Ensmenger, 2012). Confining women to low-level careers in computing created a gendered division of labor and feminized lower-level positions such that women's contributions to the field became concealed (McGee, 2018). It is important to note that the women who pioneered entry into the computer science field were privileged by a class status that allowed for unique opportunities in higher education and employment (Carr et al., 2018).

Tensions between subject and community. In order to further understand women's persistent underrepresentation in computer science, some literature focuses on women's experiences within and perceptions of IT culture. Hewlett (2008) studied professional women in the science, engineering, and technology (SET) fields, and identified characteristics of SET culture that women perceived to contribute to their leaving the field. Similar to the characteristics of IT culture, SET culture is typified by isolationism, extreme pressure at work and unapologetically long work weeks, lack of knowledge regarding career paths, and hostile masculine cultures.

Chilly climates. While much of the literature refers to the barriers for women that are created by the masculine culture of computing, Roldan, Soe, and Yakura (2004) describe the atmosphere in which women work as a chilly climate. Interestingly, the characteristics of a chilly climate align with the characteristics of IT culture, including extended work schedules, rewarding innovation over teamwork, and a double standard where men are rewarded for assertiveness but women are punished. Other elements of chilly climates include references to esoteric cultural knowledge, sexually explicit references that represent women as subdominant, exclusion from conversations and being ignored, and low numbers of women (Cheryan, Plaut, Davies, & Steele, 2009; Faulkner, 2009; Kanter, 2006; Turco, 2010). Women's interest and sense of belonging in the field were lowered by geek-culture images saturating the workplace (Cheryan et al., 2009) and media in the workplace that problematically stereotypes gender (Davies, Spencer, Quinn, & Gerhardstein, 2002). Men were not affected by images in the physical workplace.

Values and myths. The difference between men and women's participation in STEM fields can be attributed to the way in which men and women are socialized and the subsequent influence on their moral code and beliefs (Eccles, 1987). In particular, the myth that computer science is an individualistic field, and does not highlight societal impact, makes women less likely to choose computer science as a major (Weinberger, 2004; Wilson, 2002). Consequently, because women's values often do not align with the myths of the computer science field, women believe that they cannot succeed in the major nor be satisfied by a career in computer science (Beyer, Rynes, & Haller, 2004). Additionally, the common myth of the computer scientist as an isolated 'hacker' in a dark room tends to dissuade women from becoming computing majors (Cheryan et al., 2009). Women's interest in computer science is significantly more affected by the geek myth or hacker image than men's (Fisher & Margolis, 2003).

Gender constructs. Although the IT profession has been shown to be embedded in a masculine culture, feminist sociologists have argued that workplace cultures are not gender-neutral since the 1970's. Kanter (1977) demonstrated that a woman's primary barrier in a workplace is due to underrepresentation in positions of power. Furthermore, Kanter (1977) found masculinity to be embedded within the top positions of an organization despite reporting sex-neutrality. Kanter's work solidified the notion that gender organizes the structure of organizations. At this point, the question arises regarding whether the IT field is just a microcosm of a larger gender issue in organizations, or whether the IT field is impacted by gender organization, and furthered by its own unique characteristics such as the focus on technology.

Gender and technology. Along these lines, feminist literature raises the concern that technology has historically and culturally been described by masculine norms (Wajcman, 2009b). For example, plugs and cords are described by whether they are inserted (male) or whether they accept the insertion (female). Likewise, this research describes the manner in which business structures and leadership positions in IT were shaped by masculine norms, creating barriers for women to rise through the ranks (e.g., Wajcman, 2009; Wajcman & Martin, 2002). Interestingly, women have recently been able to support each other by transforming spaces and norms using newer technologies (e.g., Haraway, 2000). Although these transformations are possible, they often reproduce traditional power systems, but in new spaces (Wajcman, 2009b).

Tokenism. Some researchers explain women's status in computer science by an occurrence that Kanter (1977) attributes to proportional representation. Kanter argued that when 85% of people in a group share a common characteristic such as sex, ethnicity, or religion, then the remaining 15% of people are *tokens*. Kanter contends that tokens suffer from: increased pressure due to increased visibility; isolation resulting from dominant persons exaggerating differences between themselves and tokens; and diminished statuses within the organization resulting from stereotyping and image flattening (Kanter, 1977; Sackett, DuBois, & Noe, 1991).

Zimmer (1988) and Yoder (1991, 1994) argue that tokenism does not adequately explain the whole story, as it ignores systemic gender and racial discrimination. For example, tokenism does not account for the difference in power and social status between white men and people of color. Tokenism also does not consider the negative and extreme reactions of the dominant group when the number of tokens increase in an organization (Yoder, 1991). For women, this has significant effects, as it could lead to increased occurrences of sexual harassment, exclusion, and aggression.

Summary. The occupational culture of computing is grounded in the gendered professionalization of the field, and furthers male toxicity by emphasizing misogyny, intensity toward technology, and male-centric interests. In essence, the culture of computing emits and is entrenched in white male culture. Overall, the literature suggests that women and people of color in computer science work in repressive cultures that also halt their advancement in the field. Despite interventions to feminize the field or empower women and people of color with grit to withstand the toxic culture, the computer science field remains dominated by white men.

Mediating artifacts. Mediating artifacts are the symbols and tools used by a community in a practice. They are socially constructed over time through practice. As such, mediating artifacts are the product of a historical-cultural process. Artifacts are not static; rather they are continuously shaped and reshaped by the community to meet the needs of the community engaged in some practice (Bardram, 1997). Programming requires the use of programming languages and environments; however, software development requires much more than writing code (Naur, 2007). Software development requires collaboration, ideation, and documentation for others to read. Tools like project management software and communication channels have been developed over time by the community for the community to continue the practice of programming. Finally, programmer knowledge is an important part of programming that even transcends documentation in that: it contextualizes code to the problem statement of the organization; it structures code cognitively; and it allows for a two-way channel between the code and the real-world context in which the code is being created (Naur, 2007).

Inherent bias of technology. Tools mediate human activity as well as other tools used in praxis. In other words, a technological product can be used to develop further products, as in the case of communication software supporting future tech development, and tools used in one context can be implemented in different ways with some unrelated activity (Spinuzzi, 2011). Even though technologies can be used to in the development of other technologies, ultimately, they have all been programmed by humans. As such, technologies that operate with some autonomy such as sensors that monitor and collect data or machine-learning programs that continuously learn new information are inherently limited by human design (Leonardi, 2011). Tools, then, are produced through a social process, and are embedded in the norms and culture of the organization of their origin and development (Blunden, 2010). In other words, humans continuously develop technology while in turn being shaped by those technologies (Hyysalo, 2010; Kuuti, 1996). Subsequently, technology, and by extension software, are not neutral; rather, they are shaped by and contain the biases of actors within the system. This is particular concerning given that IT lacks diversity, and the biases inherent within technology are not representative of all populations who use the technology.

Exclusive technology. The lack of women in IT is more problematic than low numbers, as a deficit of women on technology development teams means that technology often suffers from not realizing the needs of women during the development process. For example, tech assistants like Cortana and Siri were criticized for not responding appropriately to situations that primarily affect women such as sexual assault (Miner et al., 2016). Similarly, Apple's proprietary health app faced criticism in 2014 for not supporting the ability to track women's menstrual cycles (Alba, 2015). Some critics argue that these failures reveal the masculine culture of IT and the reason underrepresentation in IT is a global issue (Chemaly, 2016).

The male-centered culture of programming produces technology that fails to produce products that meet women's needs, and the white-centered culture of programming produces technology that suffers from inattention to the needs of people of color. For example, facial recognition software often fails to recognize the faces of people of color, which could be a consequence of the disproportionate numbers of white developers on development teams (Breland, 2017). Furthermore, the industry's reliance on pre-existing libraries of code is problematic in that code repositories are embedded in the cultures of homogenous groups, and will continue to fail to recognize the needs of those outside of that group. Lack of diversity in software development means that the needs of those not represented in the field are also not represented in the technology that those engineers produce. As algorithms and software assume a larger role in society (e.g. crime prediction), it is necessary for the field to become more diverse so that the technologies that mediate any practice represent everyone.

Summary. The white-male culture of computing creates barriers for women and people of color from industry, which in turn affects the technology produced by the tech sector. There is therefore tension between the community and the social artifacts used by the community in practice. The tools and symbols developed by practitioners to develop software are embedded within the biases and culture of those who develop them, and in turn mediate the practice of newcomers to the field, describing the tension between the symbols used by the system and the subject engaged in increasing the workforce. More importantly, those technologies mediate the practices of everyone in the world who uses them, creating a tension between the community and the division of labor, as the division of labor is hierarchically structured according to the belief systems of the system. Specifically, the tools and symbols used by teachers and students in the K-12 and college system are also embedded within the white-male culture of computing. This implies that the system of activity of industry laborers affects the system of activity of those who are learning the trade in preparation for entry into the field.

The Cultural-Historical Activity of Programming in K-12 Schools

The activity of programming, rather than any programming organization or population, is embedded within an occupational cultural milieu that is exclusionary. Industry veterans' intense relationship with technology led to syntactical languages being too difficult for many introductory learners to use (Ko, Myers, & Aung, 2004). Specifically, the cultural belief that scripting languages are for real programmers while use of visual languages is not 'real' programming permeated into the K-12 space when companies such as College Board aligned curricula and standardized exams with industry-level scripting languages (Kelleher & Pausch, 2005).

Programming for learners is often rooted in high-end mathematical concepts like prime numbers that are not aligned with the interests of learners (Resnick et al., 2009). Moreover, curricula are historically focused on minute mathematical investigations rather than contributions to the betterment of humankind, which would increase the number of women (Beyer et al., 2004). The cultural artifacts that mediate the activities of K-12 computer science teachers and students are socially constructed by those within the occupational culture of coding, extending the occupational culture of coding to the activity of coding in formal K-12 spaces.

Object. The object of programming in K-12 formal spaces has shifted since the introduction of Logo in the 1970's. Initially, the value of programming in schools was a mediational use for mathematics and science education (Feurzeig et al., 1969; Harel & Papert, 1990; Papert, 1972; Solomon & Papert, 1976). In 1983, College Board introduced AP Computer Science A as a mathematics class, and used Pascal as the primary language, continuing the value of programming as a means of engaging mathematical knowledge; however, in 1999, College Board changed the language to C++, and focused the curriculum on introductory collegiate computer science content (Aspray, 2016). The initial value of learning programming as means of mediating mathematical knowledge failed to hold in schools. The object of programming changed from a tool for understanding mathematics content to a subject and content area of its own. The importance of this shift cannot be understated, since the subject and object continuously shape each other (Kuuti, 1996). As the object of the system changed from an educational tool for mathematics to a subject area of its own, the tools and symbols of the system changed in accordance.

Since Margolis, Estrella et al.'s (2008) groundbreaking book *Stuck in the Shallow End*, the object of the K-12 computer science education system shifted to increasing the number of qualified computer scientists with a larger goal of increasing diversity in the field. Interestingly, Margolis et al.'s (2008) study of Los Angeles schools' computer science offerings revealed that the current K-12 educational system, in its culture, structures, and belief systems is unable to make significant systemic changes to build the computer science pipeline. There is a lack of adequate technology, technical support, and teacher expertise to develop a wide-spread computing curriculum in the K-12 system (Margolis et al., 2008).

Considering the challenges surrounding the teaching of computer science in the literature, it is worthwhile to analyze the activity system of a K-12 teacher engaged in teaching computer science for the purposes of understanding how activity within industry mediates teacher practices and student outcomes. Furthermore, understanding the activity system of a computer science teacher could provide insights regarding K-12 teachers who do not teach computer science, but still use coding in their classrooms. Figure 3 shows the activity system triangle for a K-12 computer science teacher whose object is to increase diversity in computer science, with red lines portraying tensions between components of the system.

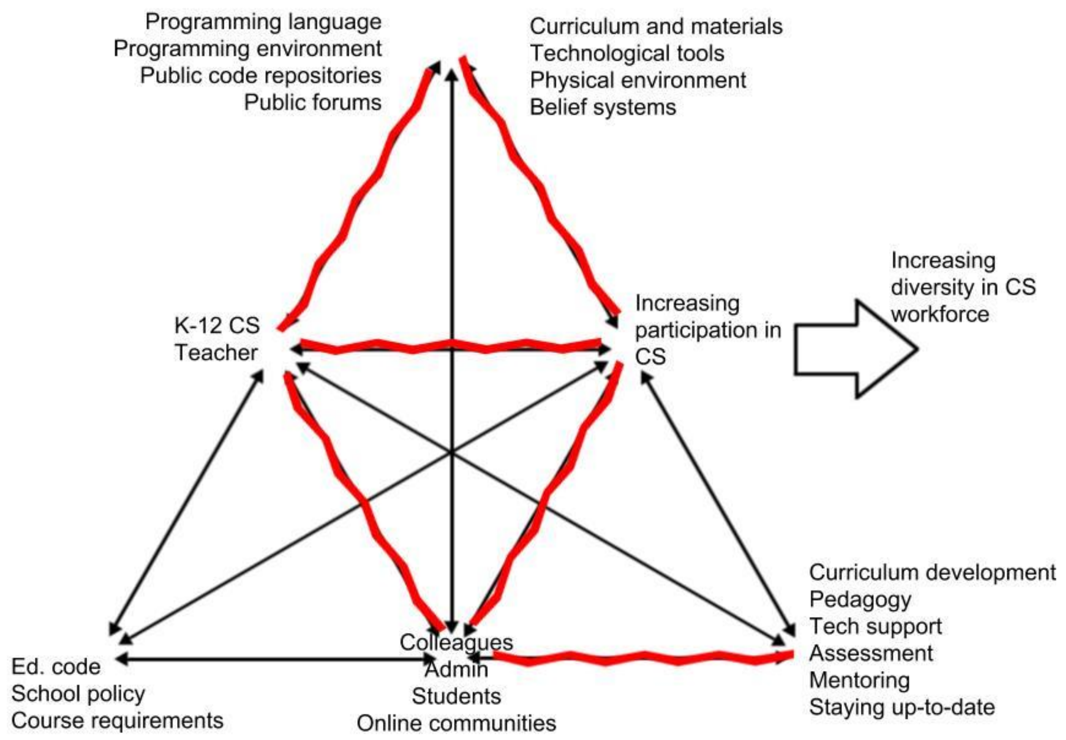


Figure 3. Activity system for a K-12 teacher to increase participation in CS.

Mediating artifacts. The most obvious tools that computer science teachers use in their craft are programming languages and environments. In order to understand how these tools mediate a teacher’s practice, it may be worthwhile to use some construct to describe the types of knowledges shared by teachers. Specifically, there is a long-standing belief throughout the teaching profession that teachers’ practice is embodied in a specialized form of knowledge. Shulman’s (1986, 1987) Pedagogical Content Knowledge (PCK) supported and shaped this belief within the teaching community. The PCK paradigm describes the different types of knowledges that teachers use and the relationships between those knowledges (Shulman, 1987). Content knowledge and pedagogical knowledge are independent of each other and yet intimately entangled sets (Loewenberg Ball, Thames, & Phelps, 2008). The PCK paradigm posits that teachers must understand what to teach (content knowledge) and how to teach (pedagogical knowledge) as well as the relationship of those two knowledges in order to optimize student learning. Neither content knowledge nor pedagogical knowledge alone is sufficient for those in the teaching profession (Settlage, 2013); rather, expert teachers blend together their content and pedagogical knowledge bases.

Technological, pedagogical, and content knowledge (TPACK) builds on the knowledge bases that teachers use to also include technological knowledge. Koehler and Mishra (2005) assert that technology is “a knowledge system that comes with its own biases, and affordances that make some technologies more applicable in some situations than others” (p. 132). TPACK is defined as the relationships and interactions between content knowledge, technological knowledge, pedagogical knowledge, and the resulting domain yielded by the combination of all three (Mishra & Koehler, 2006). Since technology is a central aspect of computer science education, the means in which teachers use technology in combination with content knowledge and pedagogy has a strong impact on students, specifically underrepresented students.

Programming languages. Although a programming language (PL) can be interpreted as a formal grammatical system built on a computational circuit (Turner, 2013), the only PL that speaks directly to the circuit is Binary, the most esoteric and rarely used language by humans (Sammet, 1972). PLs are described by the degree of translation of the source code from the conversion to Assembly. Assembly languages, those that are translated directly to Binary, are considered *low-level languages*, while Python is a *high-level language* because its source code is translated to Java, which is then further translated by lower-level languages until it is finally translated to Binary (Marowka, 2018). Programming languages implement abstraction, the design philosophy that guides the ability to write code that is meaningful to humans without considering the layers of translation and conversion occurring at sub-levels. The grammatical system that defines a PL is then directly related to its level of abstraction; lower-level languages are easier for a computer to understand, and higher-level languages are easier for a human to understand. Abstraction makes source code more meaningful for humans as they communicate with the computer. Interestingly, industry-standard and collegiate first-year programming languages such as C++ and Java require excess code to setup documents that have no effect on procedure-level code, creating barriers for new learners (Tsukamoto et al., 2016).

Programming languages are developed for specific purposes. Programmers write code in order to execute instructions that are too difficult or laborious to perform by humans (Turner, 2013). The purpose of the language defines its syntax and grammatical rules, and provides a set of tools for problem-solving within the domain of the purpose. For example, IPL-V was the first language to process lists, and afforded programmers with a means of easily creating, arranging, and modifying lists (Sammet, 1972). PLs must then be interpreted as a duality of functional and structural properties that complement each other as well as within the context in which they were developed; however, programming languages do not exist in a vacuum. PLs require other technologies to execute the code; therefore, PLs must also be analyzed as a component of a larger technological context (Pea & Kurland, 1984).

Smith, Cypher, and Schmucker (1996) note reasons why programming languages create barriers for students new to programming: learning a new programming language is as difficult as learning a foreign language; and unlike foreign languages, which are natural languages, programming languages are artificial, building on data structures and logic. Furthermore, the activity of programming requires knowledge of programming languages and their environments. Anderson, Norman, and Draper (1988) theorize that the only way to make a system easier to understand for the user is to decrease the distance between the system and the user. In terms of learning a programming language, this translates to either teaching students to think at the sub-levels of computers or to apply abstractions in programming environments and languages to allow the computer to accept inputs within the domain of human communication (Smith et al., 1996). Programming languages designed for K-12 students have taken both sides of this argument. More importantly, most of the tools required for teachers to integrate computer programming in the classroom were written by those in industry. As such, the values and biases passed to the tools are used by teachers and students to mediate their practice.

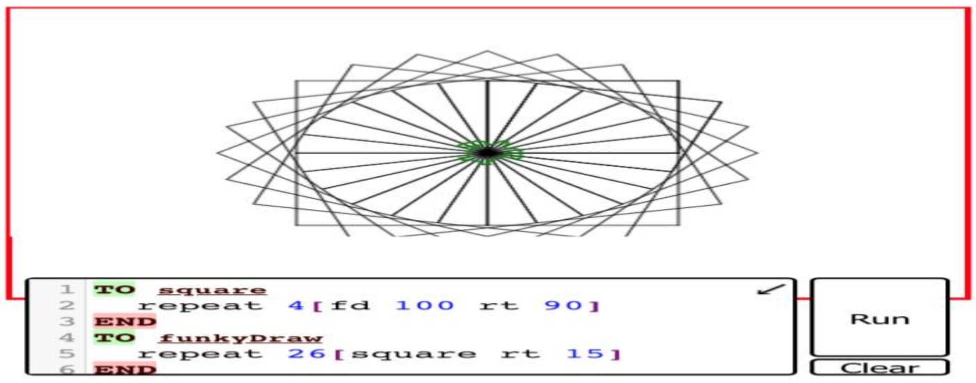
Computational thinking. In order to move users closer to the system, users must become familiar in computational thinking (CT), which implies that teachers must integrate computational thinking into their lessons. Although computational thinking is focused on the process of computing and thinking like a computer scientist, CT is more than just programming (Wing, 2006). CT is the set of problem-solving strategies employed by computer scientists to solve real world problems. Until recently, researchers focused their inquiries in computer science on the means in which programming concepts such as looping, data structures, and debugging can be taught in the classroom (Linn, 1992). As such, determining which concepts are difficult for students became important to learning scientists who then developed scaffolds for learning in the form of new curricula. Recent research in computer science education is centered on contextualizing programming by creating various project spaces for students to program. Current contexts include game design, developing wearable technologies, app development, and robotics (Kafai & Burke, 2014). Contextualizing programming is rooted in the belief that situating learning to the interests and backgrounds of students will increase participation, even among those who are traditionally underrepresented.

Contextualized computing. Contextualized computing education is defined as covering a traditional computer science course using a unified system or domain area to situate the learning (Guzdial, 2010). Some researchers have contextualized introductory computer science courses in order to increase participation. Examples of curricula that have been contextualized include traditional manipulatives ("CS Unplugged," 2019), robotics (Cuéllar & Pegalajar, 2014), and Georgia Tech's Media Computation (Guzdial, 2003). Students are attracted to programs and courses that contextualize programming; rather than writing esoteric code that adds numbers in a matrix, they can build an Instagram filter, which requires the same processes. Contextualizing code also increases female participation in computer science (Rich, Perry, & Guzdial, 2004).

Building curriculum over teachers. The current research trend of contextualizing programming is a significant divergence from the focus of computer science education in the past, and is not the norm in practice. Starting with Papert's (1980) *Mindstorms*, computer science education was focused on curriculum and learning tools. As there is no credential for computer science teachers (Lambert, 2019), there are also few opportunities for computer science teachers to receive training. Currently, more stable training programs such as credentialing programs within schools of education do not offer courses in computer science-specific pedagogy, as currently exists for math and science education; rather, the role of training computer science teachers falls on private organizations and grant-funded projects. The teachers chosen by schools and districts to teach the new computer science courses typically do not come from the computer science field; they are credentialed in another subject such as math, science, art, social studies, or career technical education (Goode, 2007b).

With a limited pool of computer science educators, curriculum developers relied on the curriculum itself to teach students; however, addressing the needs of diverse students is required for addressing the disparities in access and success in CS. To deal with the pervasive educational inequities in computer science, computer science teachers need to be prepared to develop appropriate teaching strategies (Margolis et al., 2008). They should acquire the knowledge and skills necessary to broaden access and enact welcoming and inclusive pedagogical practices that support diverse students (Goode, 2007b; Margolis, Goode, Chapman, & Ryoo, 2014). Unfortunately, computer science education has relied on new programming languages and curriculum to support students without focusing on teacher training and support.

Programming languages and environments for learners. While programming languages developed over time for professional use, certain languages like Logo, Scratch, Alice, Processing, and App Lab were developed for the purpose of education. Not to be used in industry, educational programming languages offer the ability to perform complex operations with ease (Tsukamoto et al., 2016). Interestingly, educational programming languages follow common design elements not necessarily present within industry languages like Java, C++, and Swift. This means that programming languages designed for all learners remove some of the barriers present in industry-standard languages by implementing features not present in industry languages.



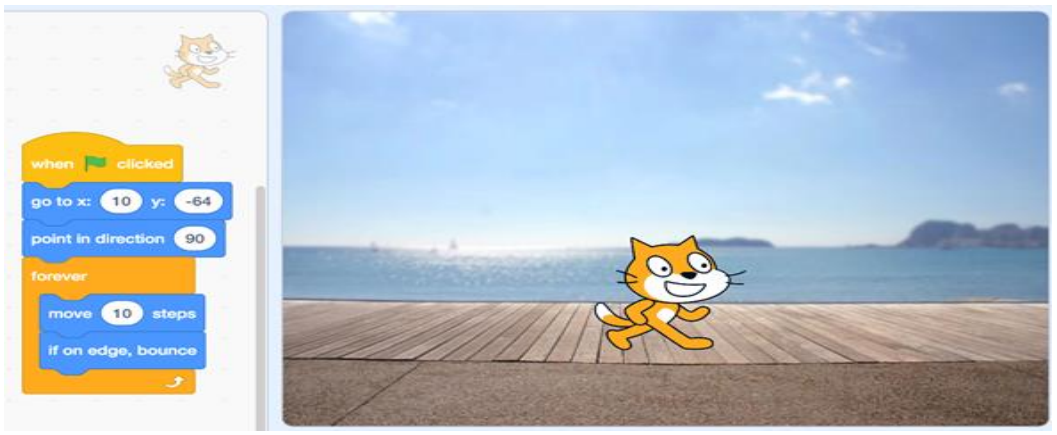
```
TO square
  repeat 4[fd 100 rt 90]
END
TO funkyDraw
  repeat 26[square rt 15]
END
```

Figure 4. Sample program in Logo.

Logo. The Logo language uses a turtle in place of a pen that users can move to draw designs using functions. Users can build their own functions using the basic functions included in the application programming interface (API). By constructing functions within functions, users can develop abstractions and build high-level designs, situated on graphical images and designs (Harel & Papert, 1990; Papert, 1972, 1980; Solomon & Papert, 1976). For example, Figure 4 shows sample code in Logo. In the example, the user developed a function called *square* that tells the turtle to move forward one hundred pixels and turn right ninety degrees, and to repeat that process four times. The function *funkyDraw* then uses the square function in its code, an example of abstraction. This function tells the turtle to draw a square and turn right fifteen degrees, and to repeat that process twenty-six times. Curriculum in Logo was centered around students developing scaffolded functions in order to increase mathematical thinking

(Solomon & Papert, 1976). The Logo language uses graphics and basic commands to mathematically teach students how to think like a computer using basic functions and abstraction in order to prepare students for more advanced languages in the future. Scaffolding functions removed the need for a trained teacher; rather, emphasized the use of curriculum aimed at independent practice.

Scratch. MIT Media Lab's Scratch was designed to appeal to youths who would not otherwise consider themselves to be programmers. The developers wanted any child, given any background or experiences, to program their own stories, simulations, animations, and interactive stories, and to share them with anyone else in the world (Resnick et al., 2009). Scratch was a response to the complaints that programming languages are too difficult for K-12 students to understand, and was designed for all learners with an emphasis on social orientation and media creation. Using the design philosophy that programming languages should have a low floor for access and a high ceiling for future opportunities of complexity, the Scratch language uses visual blocks and a community of users to support each other (Resnick et al., 2009). Furthermore, the central design philosophies for Scratch are to make programming more hands-on, more meaningful for students of various backgrounds, and more social than other languages. As such, the environment is situated on media-related content, and the sharing, remixing, and liking aspects of the community give users a non-toxic community in which they can move to expert (Resnick et al., 2009). Figure 5 shows a screenshot of a simple program that tells the cat to move to a specific position and point to the right, and continually walk. If it hits a wall, it will point in the opposite direction and continue walking. The Scratch language attempts to teach the computer inputs that are relatable for humans. In this case, the visual puzzle blocks are the relatable construct, and those blocks reveal more complex code under the hood, compiled on more advanced languages. More importantly, the social features of Scratch (e.g. sharing, remixing, and commenting) motivates students to join a community of programmers. Interestingly, teacher support and training are not a part of the Scratch project.



```
when clicked
go to x: 10 y: -64
point in direction 90
forever
    move 10 steps
    if on edge, bounce
```

Figure 5. Sample program in Scratch.

Summary. Programming languages designed for learners contain features that are not present in professional programming languages. The Logo language does not contain extraneous code for the document itself like Java and C++. Users begin programming whatever they want the turtle to do. This simplified system of coding reduces misconceptions and focuses the code on turtle actions. Furthermore, errors in code are easily determined, as the turtle will not perform the intended action of the user. Logo is still built on a system of mathematics, and the geometric and algebraic knowledge required to perform complex functions becomes a barrier for many. The Scratch language makes mathematics less integral to the functionality of the code. Designed to engage all learners, Scratch is rooted in media, and uses visual blocks as the medium for processing media. The fact that new languages and spaces were required to attract all learners speaks volumes about the languages and spaces that are used at the hobby and professional level. The need for simplified and socially-based languages like Scratch reveal how the mediating tools that K-12 students use to begin programming create barriers; furthermore, these barriers stem from the epistemic belief systems of those who developed the tools of the trade. In addition, programming languages for learners do not include support for teachers in terms of pedagogical knowledge.

Programming environments. A programming language is a set of rules and commands that are interpreted by a machine to execute operations. A programming environment, on the other hand, is the group of software, technological tools, and hardware that facilitates the writing of code. Programming environments allow programmers to edit, revise, test, import libraries, debug, and organize variables. For example, Java is a language, and Eclipse and NetBeans are programming environments where programmers can write, edit, debug, and test their Java source code. The programming environment enables communication, sharing, and collaborative tasks.

Programming tasks. Programming environments reduce the cognitive load of programming (Bruner, 1956) by distributing low-level cognitive activities across the software, allowing the programmer to focus on higher-level cognitive skills such as program design, efficiency, and readability (Pea & Kurland, 1984). Programming environments lessen the cognitive demand of programming languages, as languages must be written within an environment. For this reason, it is important to navigate the tasks required while programming, and how programming environments support those tasks.

Writing code. There are two aspects to writing code: facts and syntactical knowledge, and deep, conceptual understanding of logic. In terms of syntactical knowledge, Davis, Linn, and Clancy (1995) found that managing parentheses and quotes in LISP was difficult for students, and developed an intervention to alleviate struggle. In terms of conceptual understanding, novice programmers often do not understand the relationship of lines of code to others within a single document or across documents (Bonar, Ehrlich, Soloway, & Rubin, 1982; Soloway, Ehrlich, &

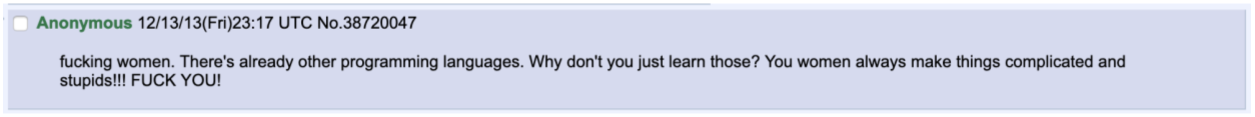
Bonar, 1982). In addition, novice programmers struggle with control of flow, which requires tracing loops and conditionals (Mayer, 1976; Miller, 1974; Sime, Arblaster, & Green, 1977). The programming environment can alert the user to compilation errors and control of flow challenges while the user writes code, enabling the user to focus on the larger structure and flow of their code.

Tech neutrality. Technology, including code, is not neutral. While computers interpret code execution in unambiguous, non-contextual means, humans write code within their experiences and personal epistemologies (Bowers, 1997). For example, a computer can process a boolean variable as true or false, but does not understand the context of that variable. Likewise, a human can choose to code a boolean variable to represent gender, inserting the programmer's assumption that gender is binary, and thus negatively affecting users who might not identify within that construct. Unlike natural phenomena, humans do not discover technology; rather technology is constructed. Computational artifacts are made for and by people, and incorporate the cultural knowledge of its developers. For example, an artificial intelligence engine was built to judge a beauty contest, and lowered the scores of contestants with darker skins tones (Vanderborght, 2018). As such, the socio-ethical side of technology and innovation becomes increasingly important. Consequently, the masculine culture of computing is embedded within the value of technological innovations. This leads to two consequences: the world is mediated by technology with white masculine value and; increasing diversity in the computer science field will increase the diverse perspectives that innovate.

Software informalisms. Scacchi (2002) referred to the importance of software informalisms, which are the symbols and tools that programmers use to structure, document, communicate, and build code. In other words, software informalisms are socially accepted rules and norms for the development of software, and are continuously evolving within online resources and documentation for the community. These resources encompass a continuously evolving distributed knowledge base as programmers gain knowledge and add to the resources (Scacchi, 2002). These resources and symbols then become an extension of practitioners in computer science (McLuhan, 2006), and mediate their practice, including the development of code and sharing of practices.

An equity challenge arises when the information resources and artifacts that programmers use for software development are written by a single demographic. Martin (2015) found that men represent 80% of commenters in online spaces such as news sites. There is also a gender imbalance in high engagement sites, where women make up at most 35% and as low as 3% of users. When online forums and repositories are interpreted as digital spaces, the tendency for men to fill up spaces is called ‘digital manspreading,’ and is a spatial example of online misogyny (Easter, 2018). Digital manspreading takes space away from women, and therefore affects women’s participation within online spaces. When online spaces are an integral part of practice, as in the case of programming, digital manspreading is a much larger concern. Stack Overflow, a popular site for code repositories and forums is overwhelmingly comprised of men (Lin & Serebrenik, 2016; Vasilescu, Capiluppi, & Serebrenik, 2014). Digital manspreading within online coding repositories has given rise to the construct of the brogrammer, men who take space away from women by using the space themselves (Kumar, 2014). The gender inequity in online coding spaces is problematic, as diversity yields greater productivity (Vasilescu et al., 2015). Furthermore, the evolved and evolving context and tools of the programmer are mediated by spaces and tools usurped by brogrammers.

The following serves as an example of an event where women were displaced from programming forums used by both learners and professionals. In December 2013, Arielle Schlesinger blogged on the Humanities, Arts, Science, and Technology Alliance and Collaboratory website, asking what a feminist programming language would be. The next day, an answer from the Feminist Software Foundation was posted on message boards and social media. A play on C++, C+= (pronounced cee plus equality) was designed to be a feminist programming language, "created to smash the toxic Patriarchy that is inherent in and that permeates all current computer programming languages" (TheFeministSoftwareFoundation, 2014, para. 4). Initially, casual users believed C+= was a genuine project and that the Feminist Software Foundation was honest and had good intentions. Eventually it came to light that the Feminist Software Foundation was a misogynistic and collaborative undertaking by members of the Technology (/g/) and Politically Incorrect (/pol/) forums on 4chan, an image board renown for creating offensive content. The authors of C+= developed the documentation to mock feminism and specifically harass Schlesinger.


Anonymous 12/13/13(Fri)23:17 UTC No.38720047

fucking women. There's already other programming languages. Why don't you just learn those? You women always make things complicated and stupids!!! FUCK YOU!

Figure 6. Screenshot of response under Reddit thread about Schlesinger.

Members of the Feminist Software Foundation and others in the know began harassing users on the /g thread of Reddit, as shown in Figure 6. In the post, the anonymous responder writes, "There's already other programming languages. Why don't you just learn those? You women always make things complicated and stupids [sic]!!! FUCK YOU!" The situation was compounded by harassers pretending to be women as well as those sympathetic to the feminist movement in order to strip all others of voice in the thread. C+= demonstrates a context in which a programming language, the documentation for the language, example codes written in the language, and conversations in online forums regarding the language purposefully removed women from a digital space designed for programmers and learners.

While events like the C+= harassment are not every day occurrences, they are products of the masculine culture of computing that uses digital manspreading as a means of removing women and people of color from spaces where knowledge is shared and identities are formed. Furthermore, contrary to the myth that computer science is a solo activity, online forums and repositories are an integral part of the social and collaborative practice of software development (Storey, Zagalsky, Filho, Singer, & German, 2017). Novices in the community, such as most K-12 students engaged in a coding project for school, move from peripheral participant to expert over time within these online communities (Lave & Wenger, 1991). For this participatory culture to be rooted in misogyny and racism creates a barrier for women and people of color. When K-12 students access information and code from online code repositories and forums, they are exposed to the toxic culture and decide whether or not to continue learning how to program. The fact that new programming languages and spaces are currently being designed specifically for learners speaks to this hostility.

Summary. Without a systemic means of accrediting computer science teachers, the educational system relies on curriculum and digital learning environments to support students in their learning, creating tension between the rules of the system and the artifacts of the system. Early educational programming languages focused on mathematical thinking and supporting mathematics learning using scaffolded assignments that did not require the support of a teacher. As computer science became a subject in itself and research focused on engaging underrepresented students, more recent languages like Scratch embedded social learning opportunities and visual cues for constructing code; however, once students move beyond Scratch to a more professional-level language like Java, they are now a part of the community that relies on online forums and digital spaces for active participation, creating tension between the community and the tools of the system.

Furthermore, educational languages such as Logo and Scratch were not perceived as 'real' languages by those in the masculine culture of coding. Most schools do not offer computer science, and those that do only offer AP Computer Science A (Margolis, Estrella et al., 2008), which teaches Java, which requires the use of online repositories and forums when students become confused. Classrooms that use professional languages are then impacted by the toxic system that developed those languages, creating tension between the symbols, community, and division of labor.

The Cultural-Historical Activity of Recontextualizing Programming

Scholars have sought to address underrepresentation in computer science using varied interventionist approaches. Some scholars developed strategies that women could use to better fit in to the male-dominated field (Hellens, Nielsen, & Trauth, 2001; Hellens, Pringle, Nielsen, & Greenhill, 2000; Trauth, 2002; Trauth, Quesenberry, & Morgan, 2004). For example, Hellen, Nielsen, and Trauth (2001) interviewed twenty-two women and one man in the Australian IT sector to investigate how masculinity is perceived by women. The authors recommend a mentoring program in order to change young women's negative perceptions about the IT sector for the purpose of increasing women's participation in IT. Trauth et al. (2004) interviewed 23 women in the IT field in the United States using gender difference theory as the basis for their inquiry. Their recommendations based on the interviews were to increase mentoring specific to women, early opportunities for women to code, and management differentiation styles specifically for women.

Other scholars look at the domain of computing, and creating interventions in order to make the community more accepting for women. For example, Webster (2014) inspects the social relations between men and women and how those relationships shape the development of technology, and how those technologies in turn shape the relations between genders within software development communities. Webster recommends ending the gendered division of labor within software companies, as hierarchical status affects the relations between men and women, and mediates the development of new technologies.

Other scholars such as Spender (1997) analyze the skills and spaces required in modern software development teams in order to redefine gender roles within the community. Before the prevalence of digital manspreading, Spender predicted that the groups of women would occupy digital spaces, and as software development teams relied more heavily on digital spaces, women's increased presence within those spaces would transfer to an increase of women on programming teams. Other scholars portrayed software development in terms of 21st century learning skills, where the hard engineering skills favored by men in the past would yield to more non-technical skills such as communication and management, favoring women over time (Dahlbom & Mathiassen, 1997; Goles, Hawk, & Kaiser, 2009). Variations of this argument have been used to attempt to increase women's roles in software development teams, citing that women's interest in bettering communities and user-experience will add strength development teams (Lagesen, 2007, 2008, 2012, 2016; Woodfield, 2002). Despite interventions at both the K-12, collegiate, and professional levels, women remain underrepresented in computer science.

Margolis et al. (2008) paint a picture of a K-12 educational system that is not fit to handle broad computer science curricula due to a lack of teachers, technology, technological support, deficit systemic belief systems regarding race and gender, and state and federal mandated legislation that places increased attention in other areas of education. Despite the need for more well-trained professionals entering the IT field and a generation of students who purport to want to pursue a career in computer science (Margolis et al., 2008), the current K-12 educational system is ill-prepared to provide adequate access, specifically for underrepresented youths. In order to increase access for all, computer science education must begin in Kindergarten at all schools and continue until college, but this broadening of participation can only be done sustainably (Wolz et al., 2010), and as noted, inserting computer science as a separate class is not sustainable given the current K-12 educational system. Computer science opportunities currently exist within extracurricular spaces such as summer camps and after-school programs; however, access to those programs is privileged, and has traditionally favored affluent white men (Goode et al., 2006; Simpson, 2001).

While the robotics teams and classes currently exist as curricular entities in some schools, robotics teams and classes are not appealing to a widespread population, and are often homogenous groups (Wolz et al., 2010). Increasing participation in computing requires a sustainable and broad endeavor that accepts the premise that all students can engage in computational thinking and deserve to do so. Given the current climate of K-12 education and its systemic belief systems, this requires moving computer science away from STEM fields, which are often lacking in diversity, and finding applications for computing within non-computational fields. Furthermore, it requires thinking beyond just the curriculum and focusing on the cultures of classrooms and the communities that engage in practice in K-12 spaces (Pulimood & Wolz, 2008; Wolz & Pulimood, 2007). The remainder of this chapter provides an alternative to increase participation in computing: to intentionally repurpose computational thinking and programming into traditional, non-computational K-12 curricula, or to recontextualize coding into domains other than computer science.

Object. For this study, the object of the activity system was increasing participation in computer science. The definition of and attitudes toward digital literacy have changed since its inception in educational research. Initially, digital literacy focused on individuals' understanding and creation of new media (Buckingham, 2003). Since then, the internet and the advancement and ubiquity of technology have changed the nature in which youth communicate and participate in digital and physical communities; youth consume media when browsing, but also they produce media in the form of videos, graphics, blogs, and forums (Ito et al., 2010). The constantly evolving new media space required a shift in what it means to participate in human endeavors. Jenkins, Clinton, Purushotma, Robinson, and Weigel (2006) described the skills required by youth to be a critical participants within the new media landscape, and included the development of creative designs, considerations for ethical concerns, and technical skills. Unfortunately, in practice, the K-12 curriculum for technical skills included keyboarding and Microsoft Office classes that schools and districts predominantly used to place students of color who required credits to graduate, furthering the divide between those who had access to computer science and those who did not (Margolis et al., 2008).

Digital literacy. In 1999, the National Research Council reported that due to the constantly evolving state of technology, a skills-based approach to technology would be meaningless, as technology would move beyond the skills that students learn even before they are out of school. Instead, the report called for a fluency-approach to technology, where students understand the nature of computing systems and can learn the skills required to engage in whatever practice they choose (The National Academies Press, 1999). In the literature, researchers began to move away from the concept of *computer-skills*, and favored the concept of *computational-literacy* (diSessa, 2000).

In an effort to ground the definition of computational literacy, the National Research Council broadened its definition to engineering and the cycle of engineering (The National Academies Press, 2002). Although the report moved the conversation of literacy further away from the conversation regarding access and underrepresentation due to its use of engineering, another field with significant underrepresentation, the report did make an important distinction that echoed the works of earlier scholars like Andrea diSessa and Seymour Papert. The report differentiated between 'technological literacy' and 'technological competence.' This distinction made the case that fluency with technology is not just a professional skill for future STEM workers; rather, fluency with technology is a valuable skill for all future citizens (The National Academies Press, 2002). Although the National Research Council muddied the definition of computational-literacy to the point that it gradually became extinct in literature, it did set the stage for increasing computing for all students, which in turn set the stage for contextualizing programming.

Contextualizing programming. Guzdial (2003) predicted that a general education will include programming and computation with a caveat that to get there, computing courses will have to change. As such, Guzdial developed a course for non-computer science majors that grounded computational thinking in media processing. For example, a traditional computer science class would teach matrix manipulations by instantiating some numbers in the matrix, and then adding or multiplying the numbers by something else. In Guzdial's course, students change the pixel values of an image in order to create a filter. The processes are the same, but in Guzdial's version, the traditional content is contextualized within media processing. Guzdial's Computational Media course has seen a significant increase in women and students of color (Guzdial, 2003). Guzdial defines contextualizing programming as the reinterpretation of traditional computer science courses within another tangential domain such as media processing or robotics (Guzdial, 2010). Guzdial's contextualizing programming offers new learning modalities within computer science classes, which were shown to increase participation among underrepresented groups.

Recontextualizing programming. While approaches like Guzdial's modify existing computer science curriculum, a different approach is to integrate programming and computational thinking into courses already taken by a diverse set of students, which was Papert's (1980) original idea. Because many areas of study are impacted by computing, exposure to computing can be done in non-CS fields while still delivering meaningful content within the existing learning goals (Settle et al., 2013). For the purpose of this study, to recontextualize programming, a non-CS K-12 teacher engaged students in developing and personalizing computational artifacts using computational thinking and creativity, while providing space for students to construct meaningful understanding of the respective domain. It should be noted that recontextualizing programming is the activity of using computer programming in fields other than computer science. For this reason, robotics classes and introductory level computer science classes would constitute contextualizing programming rather than recontextualizing programming. For example, the *Engaging Computer Science in Traditional Education* (ECSITE) program was a five-year project starting in 2009 that integrated computer science content such as graph theory and simulations into social studies, biology, mathematics, art, and health classes.

The teachers in the program were trained during an ECSITE-specific professional development program centered on the curriculum. Teachers report increased understanding of computational topics among their students, and also reported to continue delivering the ECSITE curriculum beyond the program (Hug et al., 2012).

Computing supports other content. Although integrating computing into non-CS classes can increase students' understanding of computer science, it is equally important to determine whether computer science enhances the learning of content within the respective field. Union College enacted a campus-wide computing initiative, where twenty-eight faculty members from seventeen different departments integrated a computational concept in at least one of its courses (Settle et al., 2013). The authors do not describe how faculty became familiar with the computational concepts or collaborated to develop their new lessons with the computation added. The authors do, however, report that as a consequence of the collaboration, the college: published papers from students' research projects; increased educational software designed for high school classrooms; developed software user manuals made specifically for students; and developed new research tools. Moreover, most of the students in the altered courses reported that the computer science addition increased their understanding of the course content.

Programming and transfer. Even though computation can support the learning of other fields, it must be stated that there are no data to support the transfer of problem-solving using programming to problem solve in other domains. In a meta-analysis, Mayer, Dyck, and Vilberg (1986) analyzed multiple articles that tested the relationship between generalized problem solving and learning to program. The few studies that did show a positive relationship between programming and problem solving were based on anecdotal evidence, and were deemed unreliable. Based on the meta-analysis, the authors determined that there were no data to support the claim that knowledge gained from learning to program can transfer to some other more generalized problem-solving ability or intelligence. Similarly, Kurland, Pea, Clement, and Mawby (1986) targeted high schools students with at least two years of curricular programming experience to study the effects of learning programming on general problem solving skills. Not only did the authors not collect conclusive data to determine that learning to program can transfer to other problem-solving skills, but they also found that students had very little knowledge of computer science after two years of study. The argument that because one's problem-solving skills increased after learning to program then programming caused the increase in problem-solving falls into the fallacy: post hoc, ergo propter hoc (after this, therefore because of that). Exposure to programming does not impact problem-solving abilities; rather, it is the deep, expert understanding of programming that ultimately can support one's thinking in other fields (Urban-Lurain & Weinshank, 2000).

Curriculum-centrism. Recontextualizing programming has been shown to increase participation in computer science. At DePaul University, researchers developed a framework of examples based on the collaboration of faculty members in eight departments with the goal of integrating computational thinking into their general education classes (Perković et al., 2010). Again, the authors do not describe the manner of the collaboration or how teachers developed their modified curriculum. The article focuses on the types of questions asked and the modified projects that were developed in specific classes within the Liberal Studies program. No broad-level framework for integrating computational thinking into general education currently exists, although there is an example of K-12 teacher leaders developing curricular schema for others to use.

Recontextualizing programming in K-12. Researchers have seen the potential in modifying curriculum in K-12 classrooms to incorporate computational thinking and programming. As previously stated, the current K-12 educational system is not prepared to accommodate a national endeavor to make computing a requirement as there are too few computer science teachers and computer science is not a core subject (Cooper et al., 2010; Margolis et al., 2008). Integrating computing and programming into pre-existing courses and school structures is a means of overcoming these systemic barriers (Eglash, Bennett, O'Donnell, Jennings, & Cintorino, 2006; Form & Lewitter, 2011; Lin et al., 2009; Wolz et al., 2010). Additionally, new standards such as the Next Generation Science Standards (Brown & Concannon, 2018) and Common Core State Standards (Porter et al., 2011) specifically reference computational thinking as part of the practice of the subject. Furthermore, the recently written California State Computer Science Standards are designed for stand-alone CS classes as well as traditional K-12 courses that integrate computer science into the curriculum (California State Board of Education, 2018).

Recontextualizing programming has the potential to infuse computer science into the culture and system of K-12 education without relying on major system changes that rely on modifications to public policy. Adding new requirements for computer science would require the removal of another course or at the least the diminishment of other subjects, and integrating computing into pre-existing courses would render this concern unnecessary. Furthermore, the largest issue in computer science education, underrepresentation, has been shown to diminish by contextualizing computing, and recontextualizing programming is inherently contextualization. The remainder of this chapter will describe research in recontextualizing programming at the K-12 level.

Maker spaces. Maker activities and spaces provide a personally relevant context in which to engage students in computation while affording a hands-on, manufacturing approach to learning (Blikstein, 2013). Interestingly, the development of computer science in do-it-yourself (DIY) spaces mimics the research interests of both Logo and Scratch; early programming in DIY spaces were analogous to Logo in that fabrication labs were used for design and engineering similarly to how Logo was used to learn geometry and mathematical concepts (Blikstein, 2013). Later applications of programming in DIY spaces focused on media creation and tinkering, much like Scratch (Grimes & Fields, 2015).

E-textiles. Arduino is a computational design system in which hardware is compatible, and the software to program the hardware is reliant on the Arduino language. Whatever component one buys from Arduino will be compatible with any other, and is programmed using a common programming language. Hobbyists use Arduino to install home security cameras, timed lawn watering systems, and any other device that requires computation and sensors. The Lilypad Arduino is a kit that includes a microcontroller, LEDs, sensors, and switches that can be integrated with fabric using conductive thread (Buechley, Eisenberg, Catchen, & Crockett, 2008).

Kafai et al. (2014a) studied high school students' experiences making e-textile designs during three workshops that spanned one year. Using conductive thread and the LilyPad Arduino kit, students created light-up cards, bracelets with LEDs, a mural, and any project that requires human-touch as a sensory input. The authors then piloted the curriculum in two Exploring Computer Science classrooms (Fields, Lui, & Kafai, 2017b). The teachers in the e-textiles program were given professional development throughout the semester. In order to increase content knowledge, during these workshops, teachers were given the same projects that students would have to eventually complete in class. Becoming fluent in the new technologies of the curriculum was important for teachers to be able to troubleshoot students' projects. Furthermore, teachers collaborated to determine strategies for delivering the curriculum in an equitable manner (Fields et al., 2017b). In this way, teachers linked their content-pedagogical knowledge to the task at hand.

Digital fabrication. Blikstein (2013) describes a history teacher who wanted her students to use 3D printing and laser cutting in order to build historical monuments for renowned women in American history. The teacher had no experience with programming or digital fabrication, but was open to using alien technologies in her class. In order to acclimate herself to the future activities of students, the teacher experimented with the technologies ahead of time. By experimenting, the teacher could predict issues and bugs that students would experience, and therefore better supported her students (Blikstein, 2013).

In order to give the students a comfortable starting point, the teacher standardized the base on which students would design their monuments. This decision had unintended consequences, as the history project integrated mathematics and computation since students had to calculate the proper dimensions and ratios of their monument given the standardized base. By integrating computing in her lesson, the teacher's identity also shifted from a history teacher to a teacher who is open to supporting her students in their problem-solving, contextualized on the learning of history (Blikstein, 2013). The teacher's role in the classroom and understanding of classroom space also changed. In a digital fabrication lab, there is usually just one laser cutter due to cost, and this places a burden on division of labor and productivity, as only one group can fabricate at a time. The teacher found herself moving from group to group and managing the fabrication station rather than standing in a single spot for the most of the class (Blikstein, 2013).

Teachers who integrate computing and programming in their non-CS classes must be trained and open to solving problems unrelated to their content and reimagining the space and structures of their classroom.

Media creation. Wolz et al. (2010) worked with 8th grade English and Art teachers as well as predominantly Latin[x] and Black female students to develop a weeklong summer institute active learning in computing in journalism. In groups, teachers and students were given a topic to research on the Internet fit for a newscast, and then prepare an interview protocol and record an interview with a key person related to the topic. Teams then built the storyline of their journalistic piece, and animated that storyline using Scratch (Wolz et al., 2010). The study does not describe how the teachers learned throughout the process or how teachers interacted with students.

Teacher support. Barr and Stephenson (2012) gathered a group of K-12 teacher leaders who had interest in CT or expertise in curriculum development and were leaders in their communities. The teachers were asked to define activities embodied by computational thinking, and to develop projects and learning experiences for classrooms based on those activities. Again, the article focuses on curriculum development rather than teacher training and support; however, interestingly, Barr and Stephen (2012) provide suggestions to make systemic change regarding computational thinking in general education: professional development opportunities for K-12 teachers to understand what computational thinking is and how to integrate it into their curriculum; convince administrators to incentivize modifying curriculum; support teachers in their change using materials and resources; support teachers in their change using learning communities, peer learning, and exposure to the technology sector; make available open-source tools for teachers to share resources and support each other; convince current teachers why this is important; and provide more computational thinking workshops and discussions at conferences that teachers attend. The literature defines a need for recontextualizing programming: teacher training and support.

Summary. There exists minimal literature regarding teachers' experiences and perceptions while recontextualizing code, and the activity system described here will be based on what literature does exist. The purpose of this study is to re-evaluate and reconstruct the activity system of a K-12 teacher who recontextualizes code into a non-CS domain. Nonetheless, as shown in Figure 7, the information provided by the literature regarding the activity system of a recontextualizer is limited and fails to note the impact of symbols and tools on teachers' practice.

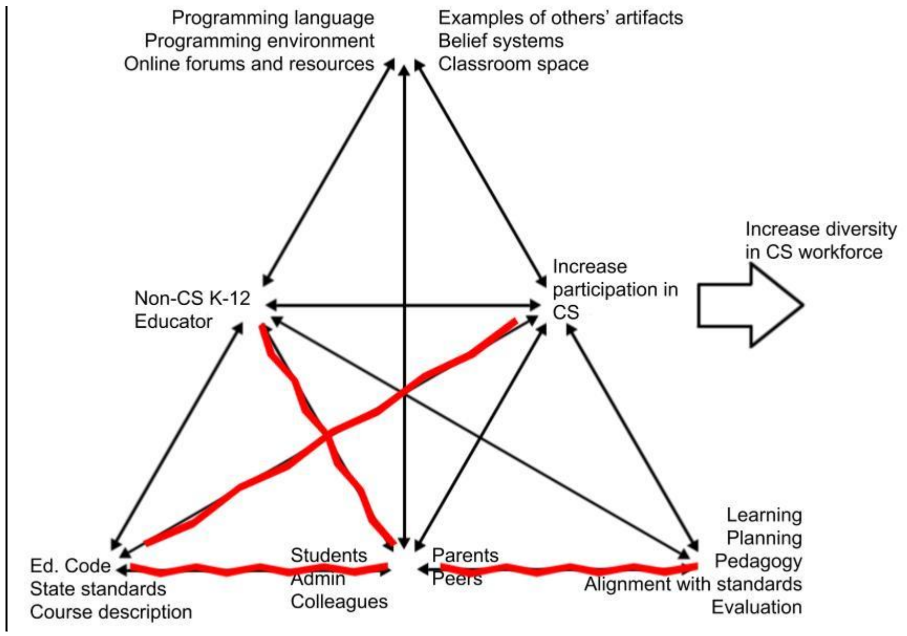


Figure 7. Activity system for non-CS K-12 teacher to increase participation in CS.

Researchers have seen the potential in modifying curriculum in K-12 classrooms to incorporate computational thinking and programming. As previously stated, the current K-12 educational system is not prepared to accommodate a national endeavor to make computing a requirement as there are too few computer science teachers and computer science is not a core subject, creating tension between the division of labor, the community, the rules of the system, and the object of increasing participation in computer science (Cooper et al., 2010; Margolis et al., 2008). Integrating computing and programming into pre-existing courses and school structures is a means of overcoming these systemic barriers (Eglash et al., 2006; Form & Lewitter, 2011; Lin et al., 2009; Wolz et al., 2010). Additionally, new standards such as the Next Generation Science Standards (Brown & Concannon, 2018) and Common Core State Standards (Porter et al., 2011) specifically reference computational thinking as part of the practice of the subject. Furthermore, the recently written California State Computer Science Standards are designed for stand-alone CS classes as well as non-CS K-12 courses that integrate computer science into the curriculum (California State Board of Education, 2018).

Recontextualizing programming has been shown to increase participation (Guzdial, 2003), as content is situated on the interests of students and relate to real-world activities. Despite studies in which curriculum is developed for non-CS teachers who integrated computing and programming in their classrooms, there is little literature regarding the needs of teachers as they make the change. In terms of the activity system of a non-CS teacher working to increase participation, there is no information in the literature about how artifacts of the community mediate the subject's practice or about the relationship between the subject and the community. The purpose of this phenomenological study was to describe the perspectives and lived experiences of teachers who have made the effort to recontextualize programming within non-CS subject matter.

This study aimed to provide insights into how teachers develop the content and pedagogical knowledge required to integrate programming and computational thinking into their respective classrooms. There are three key populations for which this study holds significance: schools of education that want to support 21st century learning; K-12 teachers who are interested in using programming in their class but lack the tools and means of doing so; and future students who will begin to view computing as a part of 21st century skills rather than an elite subject.

Although teacher preparation programs continue to support 21st century thinking and learning, they do not yet focus on computational thinking and programming across the curriculum. This study could provide information for new content for and ways of imagining teacher preparation programs to initiate teachers to the possibility of integrating programming in their classrooms as well as provide support and direction. Results from this study could also persuade departments of education to develop a secondary certification in computational thinking and programming, and could inform credential programs on best practices for supporting teachers who wish to integrate computing into their classroom.

Currently, K-12 teachers who wish to begin integrating computational thinking into their classrooms rely on the Next Generation Science Standards and Common Core State Standards, both of which mention the importance of computational thinking (Brown & Concannon, 2018; Porter et al., 2011). Additionally, the new California Computer Science Standards (California State Board of Education, 2018) are designed to be implemented in general education classes for teachers who wish to incorporate computer science in their classrooms. Although standards support content development, they do not give teachers information on pedagogical knowledge, how to manage technology, how to design the classroom space, how to manage students, or where to start looking for ideas. By describing the lived experiences of K-12 teachers who have integrated programming in their classrooms, this study aimed to provide a framework for other teachers who wish to do the same.

This study also positioned computer science as an invaluable skill that can be used within any other field. By positioning computer science as a tool for practice rather than a separate and segregated field, students in non-computational classes can engage in computing and computational thinking, contextualized on the practices and content of the respective field. This has immense implications for providing access for all to computer science, and changes how computer science is considered in the literature.

Chapter Three: Methodology

There exist K-12 teachers in fields other than computer science who integrate computer programming and computational thinking into their classes. These general education teachers integrate computer science topics such as computer programming and computational thinking into classes that traditionally have been taught in a separate computer science class, but are now contextualized within an English, history, or mathematics class.

For the purposes of this study, recontextualizing programming was defined as contextualizing programming within a non-CS space. To recontextualize programming, a non-CS K-12 teacher engaged students in developing and personalizing computational artifacts using computational thinking and creativity, while providing space for students to construct meaningful understanding of the respective domain. This study served to understand the perspectives and experiences of K-12 educators who have intentionally recontextualized programming.

The purpose of this phenomenological study was to describe the perspectives and lived experiences of teachers who have made the effort to recontextualize programming within non-CS subject matter. This study aimed to provide insights into how teachers develop the content and pedagogical knowledge required to integrate programming and computational thinking into their respective classrooms. There are three key populations for which this study holds significance: schools of education that want to support 21st century learning; K-12 teachers who are interested in using programming in their class but lack the tools and means of doing so; and future students who will begin to view computing as a part of 21st century skills rather than an elite subject. At this stage in the research, recontextualizing programming was defined as challenging students to use programming as a tool to creatively develop and personalize computational artifacts within a non-CS field.

Research Question

How do K-12 teachers describe their experiences and perceive their efforts to redefine programming in a curricular landscape other than a computer science class?

Research Design

This study used a phenomenological design. To understand the perspectives and experiences of teachers who intentionally recontextualize programming, a phenomenological approach fits best since it is grounded in the lived experiences of members of a group that share some common phenomenon (Creswell, 2018). Furthermore, the approach aims to depict the essence of the phenomenon, which in this case are non-computer science teachers who use programming and computational thinking in their relative classrooms.

Selection of Key Informants

The target population of this study was K-12 teachers in the US who have repurposed programming within their classroom for at least one year, and within a class that does not traditionally teach computer science or any course that assumes the use of computer programming. This did not include robotics teachers, computer science teachers, or teachers who work within a maker space or fab lab. This is an abstract population, as there are approximately 3.7 million K-12 teachers in the US in 2018 (National Center for Education Statistics, 2017), and not all K-12 teachers have repurposed programming within their relative subjects.

Snowball method. In order to arrive at the pool of key informants, a snowball method was used. A snowball method is useful when the target population lacks formal organization (Creswell, 2018). The researcher began the process with a small list of knowledgeable persons in computer science education who had been awarded an NSF grant for a project in computer science education. To find the initial list of knowledgeable persons, the researcher used published lists of grants on the NSF website. Principle investigators who were awarded grants specifically for increasing participation were sent an e-mail detailing the nature of the research and whether they could forward the researcher's information to any teachers or provide the researcher with the information for teachers who might recontextualize programming. Some grant-awardees provided information about forums in social media and listservs that could support the search for informants. When this happened, the researcher posted a public solicitation for informants within the given forum or listserv. Once a potential informant became a confirmed participant, the researcher asked if the participant knew of another person who fits the criteria of the study. If so, the researcher would send an e-mail to that teacher. The snowball process ended when the researcher was satisfied with the volume of data collected.

Key informant collection procedures. Initially, eight principal investigators (PIs) on the NSF website were contacted during the Summer of 2019. Each of the PIs was selected based on his/her expertise in broadening participation in computer science education. Some PIs provided social media forums, groups, and listservs that they perceived to possibly get to more teachers who fit the criteria of the study. When a potential participant was found, he/she was contacted via e-mail to assess whether he/she fitted the criteria of the study, his/her interest in being a volunteer in the study, and his/her availability. Once written agreements were made to participate in the study, meeting times were arranged via e-mail. Informed consent documents were sent to the participant via DocuSign forty-eight hours before the interview began along with a Google calendar invite for the agreed upon date and time (see Appendix A). The invitation also included a link to the Zoom conference room. One hour was allotted for each scheduled interview to allow for unplanned questions and unplanned avenues of discussion. The last participant was interviewed in July, 2019.

Challenges in finding key informants. Because many teachers take vacations at the beginning of summer, responses to e-mails were sparse during the month of June. Some grant-awardees refused to give the names of potential informants due to conflicts with their IRB protocol as well as potential interview fatigue for their participants. Some potential informants were not interviewed because during the initial e-mail process, it became evident that their conceptions of computational thinking were not in fact within the definitions presented in Chapter Two. Although eleven interviews were conducted, the researcher realized during three interviews that the subject did not meet the criteria of the study; two of the subjects were computer science teachers who contextualized coding rather than recontextualizing coding; and one was a mathematics teacher who used a computational tool that confused computational thinking for spatial reasoning. Those interviews were removed from the key subjects due to not meeting the initial criteria of the study.

Data Collection Methods

Data of two sorts were gathered to inform the study. First, key informants were interviewed about their intentions, pedagogical moves, barriers and supports for accomplishing their aims. Second, where available, teachers provided classroom examples of tasks and materials that illustrated aspects of their instructional efforts.

Semi-structured interviews. Data were collected through semi-structured interviews of eight K-12 teachers who have integrated coding into their regular class instruction for at least a year. Semi-structured interviews are a standard means of exploring meaning and perceptions in search of understanding a phenomenon. DiCicco-Bloom and Crabtree (2006) further note that “the purpose of the qualitative research interview is to contribute to a body of knowledge that is conceptual and theoretical and is based on the meanings that life experiences hold for the interviewees” (p. 314). Because this study was exploratory, semi-structured interviews were chosen to adhere to common questioning while allowing for the emergence of additional questions and themes from the dialogue between interviewer and interviewee (Creswell, 2018). Consequently, semi-structured interviews are interactive forms of communication, allowing the interviewer to navigate avenues of inquiry that present themselves in the moment. The qualitative research interview approach was deemed appropriate for this study in that it was conducive to exploring the perceptions and experiences of teachers in K-12 schools who integrate programming in their non-computational classes. Further data were collected in the form of artifacts that informants choose to share with the intention of using the artifacts to support the reporting of the informants. Artifacts were in the form of lesson plans and student work, and served to support content retrieved from the semi-structured interview.

Supporting materials. Where available, supporting materials were collected from teachers in the form of worksheets, projects, activities, and examples of student work (with identification removed). Supporting materials were gathered from four participants. The other four were unable to send examples of projects and student work. The materials were used to support the participants’ reporting of projects and student work. For example, one teacher described a project where students created robotic art to reflect their analysis of a poem, and sent videos of the finalized projects. The videos served to reinforce the type of student activity described during the interview.

Instrument Development

The semi-structured interview protocol was developed based on findings in the literature from Chapter Two (see Table 1). The interview was developed in four sections: demographic information about each respondent; rationale/intentions with the inclusion of coding; ideas or conceptions of how to go about it; and finally, the experience accomplishing this effort in the classroom.

Demographic information. Q1 asked for background on the teachers to understand how they perceived themselves as teachers within their content area, including how many years they have taught, their background in academia, and their perception of how *techie* they were. Q1 served to create a basis for their content, pedagogical, and technological knowledge to make connections to PaCK and TPaCK in later questions when they discuss their instruction and curriculum.

Rationale. Q2 asked about the source and initial reasoning for how and why the participant began to recontextualize code. Although Q5 asked about initial reasoning, it is placed at this point in the interview protocol in order to allow the participant time to reflect on their practice and craft a response based on their experiences. Q2 and Q5 were designed to allow the informant to reflect on their initial experiences in recontextualizing code, and their perceptions of how they started and why.

Ideas about recontextualizing. Q3, Q4, and Q6 asked the informant to reflect on their ideas and conceptions of integrating code into a non-computational field. Q3 asked for an example lesson in which they recontextualize code. Responses from Q3 were cross-referenced with supporting materials sent after the interview to support the type of instructional strategies and activities described. Since students and teachers face challenges while programming (Goode, 2007b; Smith et al., 1996), Q4 asked the participant to describe the challenges they experienced with using coding in the classroom and how they moved beyond those challenges. Because instructional practices, classroom setup, and student management might be different during a coding activity than a traditional activity in that classroom, Q6 asked the participants to reflect on any modifications they have made in their classroom after recontextualizing code.

Experiences accomplishing recontextualization. Q7 and Q8 asked the informants to reflect on their experiences after recontextualizing code. Q7 asked for the participants to reflect on their experiences and perceptions of coding and to provide advice for others based on their unique perspective. Q8 asked for any other information or narratives that may have been missed by the previous questions. Table 1 shows each question in the interview protocol as well as where in the literature the question originated (see Appendix B).

Table 1

Development of Interview Protocol Based on Literature from Chapter Two

Interview Protocol	Literature
Q1. Before we talk about programming, can you tell me a bit about yourself as a <subject> teacher. • How long have you taught • Is this your college/cred'l major area? • Do you consider yourself very techie? How so?	
Q2. I understand you are someone who makes use of programming, coding, with students as part of your <subject> instruction. Can you tell me how you came to do that? • Were you mandated within the dept or elsewhere? • Did you team with another teacher? • Did you get the idea from a conference or book or paper or PD? • What appealed to you about it?	Contextualizing programming (Guzdial, 2003, 2010; Rich, Perry, & Guzdial, 2004), Teacher needs (gap in literature)

(continued)

Interview Protocol	Literature
Q3. Can you give me a recent example of a time when students needed to use code for class? Can you show me the assignment or can you show me an example of student work? • How did that go? Did they need instruction on coding? Did you support that? • How much of your in-class time typically gets spent on this? Is it a one-time thing? • Are students assessed on their code work? Why / why not?	Recontextualizing programming (Pulimood & Wolz, 2008; Settle et al., 2013), Curriculum development (Barr & Stephenson, 2012)
Q4. I understand that not everything goes as planned during a lesson. Can you tell me about a time when coding became problematic or difficult in your <subject> class? • What did you do to accommodate?	Barriers while programming (Goode, 2007b; Smith et al., 1996), Teacher needs (gap in literature)
Q5. Can you tell me what prompted you to make the choice to integrate programming in your <subject> class? • As best as you can tell, have you seen changes in enrollment? • How about students' experiences? • Do you have any regrets? Or things you might change?	Teacher perceptions and experience (gap in literature), Underrepresentation (Guzdial, 2003a; Kafai & Burke, 2014; Margolis et al., 2008; Margolis & Goode, 2016)
Q6. Can you give me an example of a change you have made in your classroom since you started programming? • What prompted the change? • Do you do it every day in your class? If not, when do you use it?	PCK (Shulman, 1986, 1987), TPaCK (Mishra & Koehler, 2006)
Q7. Based on your experience, what would you tell another teacher who wanted to use programming in their classroom?	Teacher perceptions and experience (gap in literature), teacher needs (gap in literature)
Q8. I've come to the end of my prepared questions. Considering what we've discussed, is there anything else you would like to say? Have you had experiences that you would like to share that have not been shared yet?	Teacher perceptions and experience (gap in literature), teacher needs (gap in literature)

The interview protocol was designed to elicit specific responses so that a narrative of experience and perception could be built during analysis. Specifically, teachers' initial reasoning and source of recontextualizing programming, support systems and challenges faced when implementing lessons, information about the kinds of instruction and curriculum used for coding as well as assessment, perceptions of student reception, changes made in either the teacher or the culture of the classroom, and teachers' perceived value of recontextualizing programming were considered as milestones in building a narrative of their experiences.

Data Collection Procedures

Data collection was comprised of three procedures. First, key informants were sought during a two-month period during the Summer of 2019. Once a key informant was confirmed, a semi-structured interview was conducted virtually. Finally, after the audio file from the interview was transcribed to text. All data collection processes occurred between June and August, 2019.

Interview procedures. All interviews were conducted online using Zoom, a communication software that allows for video and audio recording. All interviews were conducted over a two-month period for convenience. Each interview began with a review of the purpose of the research study, reminding the interviewee that the interview was recorded and that unplanned questions might surface during the interview. The researcher then reminded the interviewee that the interviewee was at liberty to halt the interview at any time, for any reason. Furthermore, the interviewee was informed that the interview would be stopped at his/her request, and the recording erased. Once these initial acknowledgements were made, the interview and recording began.

Transcription procedures. All audio and video recordings were secured on the researcher's password-protected laptop. The audio recordings were transcribed using Microsoft Word. Prior to data analysis, informants had the opportunity to read the transcribed interview and make comments. This ensured that analyses would be grounded in the data (Marshall & Rossman, 2016).

Human Subject Considerations

Approval to conduct this study was obtained from the Pepperdine University Institutional Review Board (IRB) before the researcher made any communication with anyone targeted for participation. Upon receiving IRB approval, the recruitment process began, culminating with the distribution and collection of informed consent documents. Along with reassurance that participation in the study was voluntary, all potential subjects received detailed information regarding the purpose and nature of the study (see Appendix B and Appendix C).

Several safeguards were administered to protect the confidentiality of participants. Electronic recordings of interviews were stored on an external hard-drive with a 64-bit encryption system. The hard drive was kept in a locked safe in the principal investigator's home office. During the transcription process, pseudonyms were used for informants. A master list of real names to pseudonyms was also securely kept on the external hard-drive to protect the identities of those involved in the study. The only files containing information about the informants were the electronic interviews and e-mails between the principal investigator and participants and the master list of pseudonyms. Interviews were transcribed using pseudonyms so that identifying information was removed from all data analysis materials. Upon completion of the transcript, e-mails were downloaded to a folder, deleted from the mail server, and stored in the encrypted physical hard drive that was stored in the investigator's locked safe.

The researcher extended efforts to create a comfortable and welcoming setting during the interview protocol. In the event interviewees were fatigued or wanted to end the interview, the interviewer was willing to suspend the event and request a future date to resume.

Analysis

The textual data from the interviews were initially loaded into the HyperRESEARCH software package. A priori codes were loaded into the codes window, and the transcripts for S1 and S2 were coded in HyperRESEARCH using the a priori code tags.

A priori codes. Codes were developed based on the research question and the development of the interview protocol. The interview protocol sought to determine: demographic information about each respondent; rationale/intentions with the inclusion of coding; ideas or conceptions of how to go about it; and the experience accomplishing this effort in the classroom. Codes were designed to break down these concepts and represent informants' experiences and perceptions of each.

Informant demographics. A code for when the informants describes themselves, what they teach, how long they have been teaching, their perceptions of education, and their perceived attitude toward teaching. For example, "I have taught Chemistry for 5 years at my current location and 3 years at a previous school" would be coded as informant demographics.

Source. A code for when the informants discusses the original source of integrating code into their classrooms. The source could be a reference to a colleague, an article, or a workshop. For example, "I heard about a teacher doing something cool with coding simulations at ISTE, and tried it for myself" would be coded as source.

Programming language. The literature revealed that some programming languages are a significant barrier for learners (see Chapter Two). This code is for when informants describe the chosen programming language in their class and their perceptions of how that programming language mediates students' learning. For example, "Project GUTS uses pre-made simulations ran in Scratch, and our Chromebooks can run Scratch" would be coded as programming language.

Initial reasoning. The informants discussed their initial reasoning for integrating computing into their classroom. For example, "I wanted to move toward a more student-centered approach where students are engaging in hands-on learning as much as possible. I did a Google search, and found Project Guts" would be coded as initial reasoning.

Coding support. The informants discussed how they were supported if there was code or computational thinking they did not understand or could not do alone. For example, "There was this one piece of code I had to look up on Stackoverflow.com. Turns out it was not necessary for students to know so that was it" would be coded as coding support.

Value. The informants described what they perceived to be the value of learning to code or coding in school. For example, "Through coding, students found deeper understanding of the content of my course" would be coded as value.

Impact. The informants described how they perceive the impact on students. This could also include the informant describing how students respond to coding. For example, "Students were engaged more than usual, and spent a lot of time personalizing" would be coded as impact.

Modifications and learned lessons. The informants presented modifications they have made or will make in the future based on their experience. They could also present lessons learned from the experience. For example, "Next year, when I run this lesson again, I am going to present the coding in a different way" would be coded as modifications. Table 2 shows the a priori codes that were used.

Table 2

A Priori Coding Scheme for Data Analysis

Code	Definition	Example
• Informant Demographics	Attribute code for the informant's teaching position, years teaching.	"I have taught Chemistry for 5 years at my current location and 3 years at a previous school."
• Source	The informant discusses where they initially got the idea; could be a reference to a colleague, an article, a conference session.	"I heard about a teacher doing something cool with coding simulations at ISTE, and tried it for myself."
• Programming Language	The informant discusses which programming language and environment they chose to use in their class.	
○ Block-based	The informant references a block-based language like Scratch, Blockly, and AppLab.	"Project GUTS uses pre-made simulations ran in Scratch, and our Chromebooks can run Scratch."
○ Syntactical	The informant references a text-based language such as Javascript, Python, Java, or C.	"In my class, we use Python. I like my students using a real language used by professional coders."
• Initial Reasoning	The informant discusses their initial reasoning for integrating computing into their classroom. In Vivo coding.	"I wanted to move toward a more <u>student-centered approach</u> where students are engaging in <u>hands-on learning</u> as much as possible. I did a Google search, and found Project Guts."
• Coding Support	The informant discusses how they were supported if there was code or computational thinking they did not understand or could not do alone.	"I went to Stack Overflow, and found a thread for the challenge I had"
• Value	The informant describes what they perceive to be the value of learning to code or coding in school. In Vivo coding.	"Through coding, students found <u>deeper understanding</u> of the content of my course."
• Impact	The informant describes how they perceive the impact on students. This could also include the informant describing how students respond to coding.	"Students felt that Scratch was not real programming and many did not participate." "Students were engaged more than usual, and spent a lot of time personalizing."
• Modifications and Learned Lessons	The informant presents modifications they have made or will make in the future based on their experience. They could also present lessons learned from the experience.	"Next year, when I run this lesson again, I am going to present the coding in a different way."

Teacher support. The informants discussed how they were supported in their endeavor to recontextualize code. For example, "I got a lot of ideas and help from other teachers on Twitter" would be coded as teacher support.

Activity example. The informants gave an example of an activity they do in their class that requires coding or computational thinking.

Advice. The informant gave advice for others who want to start recontextualizing programming. For example, "I would advise others to just jump in and do it" would be coded as advice.

Assessing code. The informants described whether they assess code, and what that assessment might look like. For example, "I grade on a rubric, and give them certain functions that have to be in their project" would be coded as assessing code.

Challenges. Challenges emerged as a theme, and each of its sub-codes was emergent based on multiple informants describing similar challenges.

Administration. The informants described administration as not supporting their activity. For example, "My administration would not support the use of computers in science instruction" would be coded as admin.

Classroom management. The informant described classroom management as a burden during coding. For example, "I tried to do it, but there was too much chaos in the room" would be coded as classroom management.

Student demographics. The informants described the make-up of students in their classes as posing a challenge to their activity. For example, "I have low performing students since we are in a lower income area, and they just can't code" would be coded as student demographics.

Teacher training. The informant described a lack of teacher training for integrating code in class. For example, “I was told I had to do this, but there is no training in my area for this type of thing, and nobody at my site can help” would be coded as teacher training.

Tech gap. The informants described lack of technology at their site or other sites. For example, “I tried working with a teacher at another school, but they do not have any computers” would be coded as tech gap.

Tech issues. The informant described challenges with technology not working correctly. For example, “The WiFi goes down, the robots don’t power up, and everything goes wrong” would be coded as tech issues.

Testing culture. The informant described the school’s focus on testing and state scores as a challenge to overcome. For example, “It’s hard to find time since we have to get through all the standards for the state exam” would be coded as testing culture.

Support student code. The informant gave specific pedagogical examples for supporting students in learning how to code or overcome challenges faced while coding. For example, “So I show them the basic operations in Scratch, and then if they have any individual needs during the project, I sit with them and do it with them” would be coded as support student code.

Time. The informant described how much time during class is allotted for coding. For example, “Since these are major projects, I do them like three times a year, and we code every day during the project” would be coded as time.

Table 3 shows the final coding rubric after the a priori codes were modified from the pilot interview, and after the emergent themes were given codes during the iterative process of analysis.

Table 3

Final Coding Rubric

Code	Definition	Example
• Source	The informant discusses where they initially got the idea; could be a reference to a colleague, an article, a conference session.	"I heard about a teacher doing something cool with coding simulations at ITSE, and tried it for myself."
• Programming Language	The informant discusses which programming language and environment they chose to use in their class.	
○ Block-based	The informant references a block-based language like Scratch, Blockly, and AppLab.	"Project GUTS uses pre-made simulations ran in Scratch, and our Chromebooks can run Scratch."
○ Syntactical	The informant references a text-based language such as Javascript, Python, Java, or C.	"In my class, we use Python. I like my students using a real language used by professional coders."
• Initial Reasoning	The informant discusses their initial reasoning for integrating computing into their classroom. In Vivo coding.	"I wanted to move toward a more student-centered approach where students are engaging in hands-on learning as much as possible. I did a Google search, and found Project Guts."
• Teacher Support	The informant discusses how they were supported in their endeavor to recontextualize code.	"I got a lot of ideas and help from other teachers on Twitter."
• Value	The informant describes what they perceive to be the value of learning to code or coding in school. In Vivo coding.	"Through coding, students found deeper understanding of the content of my course."
• Impact	The informant describes how they perceive the impact on students. This could also include the informant describing how students respond to coding.	"Students felt that Scratch was not real programming and many did not participate." "Students were engaged more than usual, and spent a lot of time personalizing."
• Modifications and Learned Lessons	The informant presents modifications they will make in the future based on their experience. They could also present lessons learned from the experience.	"Next year, when I run this lesson again, I am going to present the coding in a different way."
• Activity Example	The informant gives an example of an activity they do in their class that requires coding.	"Here is an example I did in my class…"
• Advice	The informant gives advice for others who want to start recontextualizing programming.	"I would advise others to just jump in and do it."
• Assessing Code	The informant describes whether they assess code, and what that assessment might look like	"I grade on a rubric, and give them certain functions that have to be in their project."

(continued)

Code	Definition	Example
• Challenges		
• Admin	The informant describes administration as not supporting their activity.	"My admin won't let me do this…"
• Classroom Management	The informant describes classroom management as a burden during coding.	"I tried to do it, but there was too much chaos in the room."
• Student Demographics	The informant describes the make-up of students in their classed as posing a challenge to their activity.	"I have low performing students since we are in a lower income area, and they just can't code."
• Teacher Training	The informant describes a lack of teacher training for integrating code in class.	"I was told I had to do this, but there is no training in my area for this type of thing, and nobody at my site can help."
• Tech Gap	The informant describes lack of technology at their site or other sites.	"I tried working with a teacher at another school, but they do not have any computers. What can they do?"
• Tech Issues	The informant describes challenges with technology not working correctly.	"The WiFi goes down, the robots don't power up, and everything goes wrong."
• Testing Culture	The informant describes the school's focus on testing and state scores as a challenge to overcome.	"It's hard to find time since we have to get through all the standards for the state exam."
• Support Student Code	The informant gives specific pedagogical examples for supporting students in learning how to code or overcome challenges faced while coding.	"So I show them the basic operations in Scratch, and then if they have any individual needs during the project, I sit with them and do it with them."
• Time	The informant describes how much time during class is allotted for coding.	"Since these are major projects, I do them like three times a year, and we code every day during the project."

Rater reliability. To ensure reliability, the researcher individually coded the first two interviews using a priori codes. Data were read through by the principal researcher and coded using the a priori codes. The researcher then shared the transcripts and coding results from the two interviews with a colleague who reviewed the transcripts and coding results and agreed on the initial results. The researcher proceeded to code the remaining six interviews. During the coding process, new themes emerged, and previously coded transcripts were revised using the new, emergent themes. Once the remaining interviews were fully coded, the researcher once again shared the transcripts and coding results with a colleague. Upon review, the PI and colleague discussed any modifications until a consensus was reached. This process enables a future researcher to repeat the work of this study for differing analysis and to utilize this research design as a prototype model (Creswell, 2018). Furthermore, during the data analysis process, the researcher found similarities and differences across reporting to ensure different perspectives were represented.

Means to ensure study validity. Since this was an exploratory phenomenological study design, semi-structured interviews enabled the researcher to investigate unplanned responses and experiences. As such, the researcher took careful precautions to ensure that personal bias would not impact the study's validity during the process of data collection.

Researcher's Relationship with the Phenomenon

In 2010, when I took over the computer science program at the high school where I teach, we had one period of AP Computer Science A. The school's policy mandated that only high-level mathematics students could take computer science, and as a consequence, a culture of elitism and gate-keeping permeated the class. By the time I took over the program, females and students of color (and cross-sections) were not included in my roster.

Upon researching similar national trends in computer science education, including the AP exam and higher education, I was referred to a team at UCLA Center-X, who had developed a class called Exploring Computer Science and a professional development that went with it. ECS was designed to broaden the participation of African-American, Latino/a, and female students in learning computer science. In the Summer of 2013, I attended my first year of my ECS PD. In the 2013-2014 school year, I piloted three sections of Exploring Computer Science. The class was a hit. In the 2014-2015 school year, we had to hire a computer science teacher to accommodate the student demand for the class.

I began facilitating ECS professional development across California and nation-wide. The increase in demand for ECS led us to pilot subsequent classes. In the 2015-2016 school year, we piloted Computer Science Principles, one year before the AP exam for AP Computer Science Principles was introduced (2016-2017 school year). In the 2017-2018 school year, we are offering four different courses, including IB Computer Science HL, and we have almost 900 students enrolled in CS. The ratio of men to women is nearing 1:1, and the demographics of students in CS parallel the makeup of the school. I continue to fight for equity and access for all students in CS education.

In Fall 2017, I served on the California Computer Science Standards Advisory Committee, where I met many other teachers and administrators also working to expand participation in computer science. The conversation of equity and access is difficult, and over time, my thinking about these concepts has changed considerably. I feel that it is important to acknowledge my biases, and situate those biases within my experience as a computer science teacher: computer science education is currently inequitable for all students; the masculine culture of computing is more permeating than currently believed; and groups in Silicon Valley are directly opposing interventions to increase diversity in industry.

Bracketing My Relationship with the Research

Due to my experiences and perspectives as a computer science teacher with a social justice lens of the world, the implications of my relationship with the phenomenon could lead to framing questions during the interview process and bias in analysis. As this is a phenomenological study in which the relationship with the researcher and the phenomenon cannot be separated, steps were taken to protect this study from my socioemotional and cognitive biases. Namely, three methods were employed: journaling as a method of reflecting on my framing and emotional process during interviews as well as during coding to ensure that my perspectives were not skewing analysis; discussions with colleagues; and taking notes during data collection in order to document moments when my thoughts veered toward potentially biased subtexts in informant responses. Throughout the data collection and analysis process, I bracketed my relationship with the research using the aforementioned three methods because bracketing is not a single procedure designed to check a box during the process of research; rather, a continual practice of self-discovery in which socioemotional and cognitive experiences may reveal themselves (Drew, 2004). Rolls and Relf (2006) note that this self-discovery process does not have to be a solitary practice; in fact, processing with peers and colleagues can help facilitate the process of self-discovery. To this end, my chair helped to support my process, and was a second observer of my journaling and note-taking.

Chapter Four: Results

There exist K-12 teachers in fields other than computer science who integrate computer programming and computational thinking into their classes. The purpose of this phenomenological study was to describe K-12 teachers’ perspectives and experiences of intentionally recontextualizing programming. Here, recontextualizing programming is defined as challenging students to use programming as a tool to creatively develop and personalize computational artifacts within a non-CS field, such as mathematics or social studies.

To recontextualize programming, a non-CS K-12 teacher engaged students in developing and personalizing computational artifacts using computational thinking and creativity, while providing space for students to construct meaningful understanding of the respective domain.

The next section offers an overview of the teachers interviewed in this study. Following this section is the description of analyses of the coded interviews and the synthesis of findings across informants. This leads into the answer to the research study question and to additional insights arising from the research.

Key Informants Profiles

The eight key informants were diverse in their backgrounds, the subjects taught, and grade level taught. In order to provide a profile of each informant, common milestones in informants’ narratives were used in accordance with the final coding rubric. Each informant described the source of how he/she started to implement code in the classroom as well as his/her initial reasoning for how it might benefit his/her students or practice. Consequently, each profile begins with a description of his/her reported source and initial reasoning, which were coded by *Source* and *Initial Reasoning.* Subsequently, all informants described challenges that they perceived to affect their practice negatively. The challenges section synthesizes information yielded by the code *Challenges*. It should be noted that demographics of each informant is embedded within the narrative rather than explicitly illustrated based on the *demographics* code. Overall demographics of subjects was discussed in chapter three as a means of describing the sample size and demographics.

The instruction section provides information related to how the informant described his/her use of code in the classroom, and was synthesized from the codes *Activity Example*, *Support Student Code*, and *Programming Language*. It should be noted that in this section, pedagogy, instruction, and curriculum are not distinguished from each other. One of the major findings of this study was the relationship between content, pedagogy, and curriculum, which will be deeply explored in a later section. Following a description of the informants' instructional practices, it is worthwhile to explore informants' perceived value of recontextualizing coding. The perceived value sections were created by synthesizing information coded by *Value*. Finally, any future advice for other teachers is synthesized in the advice section. Building a narrative using source and reasoning, instructional practices, perceived challenges, perceived value of coding, and advice for others provides a clear depiction of each participants' experiences and perceptions of recontextualizing code.

S1. S1 is a middle school science physics and biology teacher who moved to Texas from Puerto Rico. In Puerto Rico, S1 majored in general science with a minor in environmental science. She worked in a lab, but due to hours and the dangers of working with caustic substances and raising two children, she went back to school to get her teaching credential. S1 began teaching high school physics and biology, and has been teaching for fifteen years.

Source and reasoning. When asked whether she considers herself techie, S1 replied that she considers herself very techie, as she has been working with technology since she was seven or eight years old. S1 said that she initially began using Microsoft Excel as a means of allowing physics students to create complex functions and calculations for their labs and work. In 2011, S1 was chosen to attend a professional development for teachers who might pilot AP Computer Science Principles. There, she learned the programming language Snap in order to teach *The Beauty and Joy of Computing*, offered by the University of California, Berkeley. S1 recollected that after seeing the power of Snap, she taught herself Python using a variety of free online programs such as Coursera, and began using Python as a way for students to be able to create programs where they could input data during a lab and calculate results quickly and easily. According to S1:

> There was a long process to add to the right answer. So I taught them how to assign functions in a cell. So we were doing a lab. They can just put the numbers there and voila… as I entered into our computer science principle pilots teachers I learned how to use snap. That is like a version of Scratch and then I self-taught in Python. So I start playing with Python and I showed them how to use Python.

S1 described her initial reasoning for using coding in class as a means of preparing students for computer science in college by providing some access and early successes. For example, when conducting labs in physics, S1 explained that students often have to collect data and perform complex operations quickly. She requires students to create programs in Python designed to quickly transform the input data into whatever they need for the lab before coming into the lab so that they can spend less time calculating and more time modeling and planning.

Perceived challenges. S1 described her move to Texas as a means of fleeing Puerto Rico's economic and ecological problems. When she arrived in Texas, she found a job as a fourth to eighth grade physics and biology science teacher, and described her excitement to continue her practice. She recollected that when she asked to use computers in her class, administration informed her that she would not be able to use programming in her class as she once did because programming does not increase student scores on the STAR exam, the assessment used to evaluate teachers and schools during No Child Left Behind era mandates. S1 said:

> When I arrived here to the States… the administrators say, 'OK. You've got to teach science.' And then I said that if I can incorporate more computer science or using technology in my classroom they say to me you need to forget about that. You've got to teach science. And I told them that… I was way high using technology and involved in the different process. So it was like I'm going backwards.

S1 lamented that her students did not learn during that time; they could take a test, but they had no real foundational knowledge of science, and could not apply it to their lives. Consequently, S1's reporting implied that she perceives coding to be valuable when used as a tool for students to gain deeper insights into science than studying for an exam. S1 recollected:

> They are more looking for the STAR test numbers and how these tests affect the students even in science. I realize that at the end of the year my student didn't learn nothing at all from science. They know how to answer the tests. They know how to work the steps quickly and strategies to answer the tests. But they don't know science at all.

S1 described a tension between how students and teachers are assessed and students finding deep meaning within science content. She described that administration not allowing her to code in her first years in the United States was an obstacle that she navigated by changing schools and school districts.

Instruction. When S1 did use computer science in her classroom, she described her method of instruction as starting with open ended problems such as finding the density of a rock. Once students understood the science behind density, S1 said that she would ask them to find a way to communicate to a computer or robot how to find the density of a rock. As she put it:

> For example, I'd give them like a rock and I say okay I have a rock and I need the density of this rock. How are you gonna solve it. I'm a robot and you need to program me to get the density of these so give me simple steps and easy to understand simple and accurate steps. Okay verify that is a solid. Yes it is solid. OK. Now what is it a regular shape solid or irregular shape solid. Oh it's an irregular one. So now what. Oh. Get a calibrated cylinder. So they need to chunk the information in simple three word sentences. Sentence so they can understand better the process they need to follow.

In this way, computational thinking served two purposes: to review information; and to allow students to develop computational models of their solutions to inquiries in class. S1 noted that she values process over solutions, so students' computational thinking and code were assessed as part of a larger system of reasoning and exploration. She explained, "They need to understand how I can get this answer not by oh this is this. This is a possibility as this. No no no. Figured it out. You cannot just guess the answer." S1's reporting of her assessment aligns with her perception of computational thinking as a means of engaging students in problem solving techniques; computational thinking provides methods of decomposing larger problems in sub-tasks, and forcing students to show all work represents her focus on process over product.

Perceived value. S1 reported that she did not use coding or computational thinking all the time; mainly as a method of review at the end of a unit. She recounted that on coding or computational thinking days, students did not want to exit her class; engagement and student motivation increased. She said:

> When I use computers in the classroom, they don't want to exit my classroom and they really prefer that grouping and I have is that everything and they have laptops or the cart in the class they say oh we're going to use computers today. That's good.

She told the story of a girl covered from head to toe who was not allowed to touch computers at home, and the joy she expressed to S1 at learning how to program. Due to this experience and her observations of the tech gap in the United States, S1 said that she focused her efforts on using programming to increase access for all rather than preparing students for college.

Advice. When asked what advice she has for others, S1 urged other teachers to jump in, take a class, learn a technology, and play with it in class. She said that teachers are often afraid to try new things and open themselves to change. She advised, “Jump into it. Learn how to incorporate more technology in the classroom. Enroll. Because there is a lot out there. And what they need to do is just take two or three weeks to learn this and use it.” To this end, she noted that teacher preparation and training is vital for teachers to learn programming as a tool to engage students in their respective content areas, and to do so for all learners.

S2. S2 is a 17th year teacher. She spent the first fifteen years teaching fifth grade science as the lead teacher in her department. She received her Master’s in educational technology, and moved to middle school, where she teaches a STEAM agriculture course. During this time, she was appointed by the governor of California to represent teachers on the Computer Science Strategic Implementation Committee. Next year, she will be heading back to elementary school as a vice principal to move the school towards integrating computer science and science into all of their courses.

Source and reasoning. S2 recounted that her first school adopted the SAMR model for integrating technology in the classroom. SAMR is an anagram for substitution, augmentation, modification, and redefinition, and provides a scaffolded model for how technology can redefine the student experience when integrating technology in the classroom (Caukin & Trail, 2019). Although not mandated, S2 described how the school created a culture that motivated teachers to integrate technology in the classroom. S2 recalled that she began using technology such as iMovie to engage students and embed learning within larger projects. S2 then began playing with Aurasma, an augmented reality app, where students could build informational cards to display their learning in a creative and interactive way.

S2 described her area of California as singularly focused on the main source of income, agriculture. S2 recalled that when she transitioned to middle school agricultural science, she saw the power of computer science in agriculture when she visited modern agricultural centers and noticed how robotics and computation was changing the industry. She realized that learning to code is a skill required for her students in their respective futures should they choose to remain in the valley. After researching coding languages, S2 remembered stumbling upon Scratch and Thunkable, two block-based programming languages, which in Chapter Two were described as programming languages designed for education and learning. S2 said that she continued to use programming like she did with earlier technologies, as a means of engaging students and contextualizing larger projects to situate students' learning. Specifically, S2 noted:

> I want the kids to be creating their own to show their learning through their own creations and not specific ideas. Sometimes it is with a specific app or smash two together. But just to be outside of just essays. I don't read essays. We can make movies, we can do far more things. We can code.

Unlike S1, who reported to use coding as a direct mediation of science content, S2 reported to use coding as a mechanism for students to create interactive projects that communicate their understanding of content.

Perceived challenges. When asked about challenges to her practice, S2 recounted an early experience with Aurasma where students had to use her user account to post their projects, and a student changed the password, deleting everyone's work. The company could not help, and admitted that their app is not designed for school use. Interestingly, S2 mentioned that the tech failure of Aurasma motivated S2 to build the same experience for herself, which ultimately lead to her use of Scratch and Thunkable. S2 reported that while learning how to use programming languages, she reached out to the computer science teacher at her school for support. Not only did the computer science teacher at the school not support S2, but also he verbally devalued the work that she and her students were doing as if it were not real computer science. S2 described:

> He let me know his degree was in computer science, so maybe like IT something some sort of credential that was with that and he'd come in and my students would say 'oh look!' and they'd show him what they were building. He's like, oh is that just with the blocks like [laughs]? And it was just like everything was. Like it wasn't good enough, you know.

S2's reported interaction with her computer science colleague depicts behavior embedded in the masculine culture of computing (Bartol & Aspray, 2006; Hewlett et al., 2008; Todd et al., 2005; Wentling & Thomas, 2009), as described in Chapter Two.

Instruction. S2 acknowledged that despite these barriers, she integrates programming as a tool to allow students to creatively communicate their reasoning and process. S2 noted that whether students create an app to go with their baking company while learning about cellular respiration or develop a multi-screen interactive app to communicate their understanding of photosynthesis, programming is used as a tool to engage, motivate, and contextualize student learning. Specifically, S2 said:

> I let them choose what they want as they're learning the app basics. And then they have a final app project that incorporates all of that. You know, like four sliding screens. But they get to choose topic wise. We did some for agriculture in our area or photosynthesis or whatever. So they had those skills because then they can apply it to more fun stuff. So it's not like... It crosses over really well, because then they do it for fun and they show me all the random little apps they build.

S2 reported that during times when students need help with code, she uses group-based learning and peers to support individuals as well as sits next to the student, learning with the student as they tackle the problem. When asked about assessment, S2 reported that students are given a binary rubric for the features that must be integrated into their projects, and code is often intrinsically embedded within those features.

Perceived value. S2 described students as being excited and highly engaged during coding projects until they become bored and want to move on to some other technology. S2 ascribed the increase in student engagement to the hands-on creation of something they can personalize and make their own: “So I think the engagement, they really like it. They like the hands on. Even if it's not building like sciencey wise, they're still creating something that they can personalize.” S2 recounted that when she began teaching the agriculture class, nobody in the school knew what STEAM was, and counselors randomly threw students in the class. Interestingly, there were only two girls when the class began. Since then, that number has increased to nine; however, S2 admitted that the students she gets tend to be quirky and nerdy.

S2 reported that since she started using coding in her class, the number of students going into the engineering pathway increased significantly, and the number of AP Capstone sections has increased. S2 reported to be proud that most of the students in the AP Capstone are her former students. In her words:

> So all AP courses and last year as a first year, we offered AP computer science. We had one full class in high school. This year they're expecting three full classes and most of those are mine. So like I know it's been positive for them to want to continue to explore.

Advice. When asked what advice S2 has for others, she advised others to integrate coding into their classes, and to do so with respect to content. She recommended using the new California state standards for computer science because they were written to be integrated into traditional general education classes. S2 noted that building larger projects around content reinforces knowledge construction and allows students to better internalize material for long-term memory storage:

> They personalize them. They are going to remember them far more because they're going to be spending a lot of time looking at how carbon dioxide is coming from this cell. You know, they're really focusing on that because they're trying to do the programming and they know the content that they have to do.

Once teachers become comfortable with using some technology, S2 recommended that teachers build and try new things. She reported that Twitter is an incredible community with very nice, helpful people. S2 advised that teachers get initial ideas from the Twitterverse and then run with the idea and continue to build.

S3. In a prior career to teaching, S3 worked in information technology. When his son, diagnosed with cerebral palsy, turned five or six, he realized that scheduling would become difficult so he followed his own father's footsteps and found a position in technical education at a high school in Massachusetts. S3 then attended a graduate program for special education in order to enter a field more compatible with the needs of his son. S3 moved schools, and worked as an inclusion teacher, supporting special education students within mainstream classes. This means that S3 co-teaches with a general education teacher, and supports learning for students with individualized learning plans.

Source and reasoning. As a special education co-teacher, S3 reported to spend much of his time working individually with students during class. After realizing that students were not reflecting on the soundness of their answers after using a calculator or could not use the order of operations correctly when computing, he reported to being creating little programs in Python designed to help students check their work. He said that he would distribute the programs using repl.it, and noticed that his students were amazed that programming could get answers so quickly. S3 asserted that he used this amazement as an opportunity to introduce students to code so that they could create their own programs to support their calculations in class. S3 reported that due to his background in information technology, he understands the importance of programming in any field, and uses coding as a means of reinforcing content area knowledge as well as providing access to an important skill. S3 reported:

> You want to see it as a mechanism to a means to an end, so to speak, you want the kids to be computer literate. And that used to mean how to use Microsoft Office as an example. Then it became, oh, can you write a web page. But now it's become can you actually create something using the computer as a tool.

S3 noted that computer literacy not only includes word processing and web design, but also coding in order to create, and this aligns with new models of digital literacy, as described in Chapter Two (Jenkins et al., 2006; Knobel et al., 2010).

Perceived challenges. S3 reported that he works with many teachers, and admitted that not all teachers are receptive to coding in class. When asked, S3 identified three barriers for his co-teachers to integrate coding into their respective content: “So they have to learn, you know, A, they have to have the ability to understand the program. B, they have to understand how to embed it in there in their assessment. Third, you have the equipment issue.” In other words, teachers do not feel comfortable teaching something they do not understand fully, teachers do not know how to integrate programming into their curriculum, and teachers are not open to alternative methods of assessment. Additionally, technology often fails. S3 reported to have a class set of tablets in the rooms in which he works, and the tablets are often out of power, take time to power up, the network crashes, and there are hardware issues. S3 noted that due to the technology issues, programming in class requires significant planning in advance as well as back-up plans. For this reason, S3 described programming as a special activity rather than a fully integrated part of the curriculum, as he wishes it to be. Finally, S3 reported that due to classroom constraints, he cannot spend a lot of time teaching students how to code; they need to be able to look at his examples and develop programs based on his examples. S3 observed that student level and ability become barriers when students are not given proper instruction.

Instruction. Because S3 is a co-teacher, he described being placed in specialized co-taught classes, where students with individualized learning plans are integrated into mainstream classes. He defined his role in the classroom as supporting the content area teacher by providing specialized instruction for students with special needs. S3 noted that although he may share a class with a general education teacher, that teacher may have three or four other sections that are not co-taught. S3 acknowledged his role in the co-teaching relationship, and reported to not want to provide an experience for students that vastly deviates from the teacher's other classes. In his words, "when you're a co teacher, you don't want to make your one day there very different from the rest of your days there." Consequently, S3 reported that programs designed to support students' calculations is his method of injecting himself in the learning environment without distracting from the teacher's plans, but he noted that wishes he could do more. S3 described how students learn code: by looking at his examples and replicating the code. S3 reported to having a visual that allows him to present the code and describe it line by line. In order to reduce the impact on a teacher's class, S3 noted that his coding assignments are treated like any other homework assignment, and depending on the general education teacher, are often not required for a grade.

Perceived value. When asked about impact on students, S3 acknowledged that a large percentage of the students say, "Programming is not for me." Despite his desire to increase participation among underrepresented youth, he reported to observe students entering his class with the mindset that they do not see themselves as someone who does coding. S3 noted that student identity is a broader issue and is affecting his ability to effectively engage students in integrating programming into other content areas such as chemistry and math. S3 noted:

> So there's that there's that initial problem with people saying, I don't see myself as being a person who does coding. So I think that's that that's a sort of a low level kind of thing that people think of when they think about coding. They think about Bill Gates, Mark Zuckerberg, Musk. They think of these famous people who basically look like me. Mean, you know, white men.

S3 reported to try to dispel myths that programmers are predominantly white, and gave credit to media such as *Hidden Figures* for providing role models.

Advice. When asked what advice he would give to others, S3 advised teachers who want to start integrating programming in their content area to not try too many things and to keep it simple. As previously stated, S3 reported that he wants students to see how programming is a viable tool that can be useful in any future endeavor. He acknowledged that perhaps coding will lead students to take a computer science class, but teachers should treat code as a tool and integrate it into their curriculum in a way that makes sense. S3 also advised that teachers give students a concrete platform and provide students with limited options at first, and as students increase their knowledge, add more tools to their toolbelt. S3 said:

> Don't try to do too many things. Find the one place that you know this could be successful... And then I guess KISS. Keep it simple, stupid. Don't try to tell me I can turn my students in a chemistry class to be programmers. If I can show them that programming is a useful tool to them. If I can show them that it's not that difficult to get some simple things done, maybe then they'll want to take a computer programming class.

To combat the barriers he has experienced, S3 noted that teachers should know exactly what they want their students to do, have very clear rubrics for how students do it, and work everything out in advance to anticipate any problems that may arise. Finally, S3 noted that students will make mistakes and have errors, and teachers must be able to comfort students and provide a space where failure is acceptable.

S4. S4 has taught elementary level English language arts for ten years, and in a variety of schools including public, private, Title I schools, schools designated for English languages learners, and schools with high diversity. She has a Master's in educational technology, and considers herself to be a techie person. S4 referred to herself as the person coworkers come to when they need help with technology.

Source and reasoning. S4 started dabbling with robotics in her after school clubs, and wanted to bring it to her classroom. At the same time, her husband changed career from art design to computer engineering. She joined CSTA and ISTE, and felt supported by larger communities working to engage students with technology and coding. Her background in educational technology, experience with robotics during after school programs, newfound online networks of support, and the success of her husband initiated her determination to provide exposure to computer science by integrating coding into her English class. S4 reported to initially view coding as a tool to engage students in hands-on problem-solving and making while still delivering meaningful content.

Perceived challenges. S4 reported that her challenges stem from student interest and identity. She claimed that despite her best efforts to create engaging activities, students view the coding projects as extra because it is not math or reading, which invites poor behavior during projects or not trying during a lesson. In her words:

> There have definitely been some times where I think the kids view it as an extra. You know, it's not math and it's not reading. So a lot of times kids will get kind of silly with it or they'll just not even try because they'll be like, you know, the kids that are less interested and are like, well, this is just extra. So I don't really have to try. And that was frustrating.

She acknowledged that an assessment piece could alleviate this problem. S4 reported that girls are less interesting in coding for the majority. S4 noted that after discussing the issue with her social networks, she decided to modify her projects by allowing for more choice, personalization, and creativity; to allow students to create within the scope of their interests. S4, then, acknowledged that her initial desire to make coding meaningful did not take students' backgrounds and experiences into account, and concluded that providing options and opportunities for personalization increased students' meaning-making.

Instruction. S4 reported that she begins the first quarter exposing students to as many tools as possible. Later in the year, students are allowed to be creative in their products and choose which tools they want to use. S4 described an example of a project from her class as an interactive museum exhibit:

> I tell them that they have to create an interactive museum exhibit. And the kids want to bring their content. So they have to learn the content so they choose something in our history that they want to study and then they have to create an interactive exhibit.

After learning some history, students present content in an interactive and meaningful way. Figure 8 shows an image sent by S4 of an example of what a museum exhibit in her fourth grade class might look like. She described one group's project about farming and agriculture, which focused on corn, the region's dominant revenue source. The group made a cornfield, and the user had to drive the Sphero through the cornfield; at certain points in the cornfield, new information would be presented to the user based on the position of the Sphero. When asked how frequently she uses code, S4 reported to use programming about twice per week during projects to provide engaging and interactive spaces for students to creatively demonstrate their learning. S4 noted that she assesses projects on a binary rubric, and coding is embedded within larger features that must be present in the students' products.

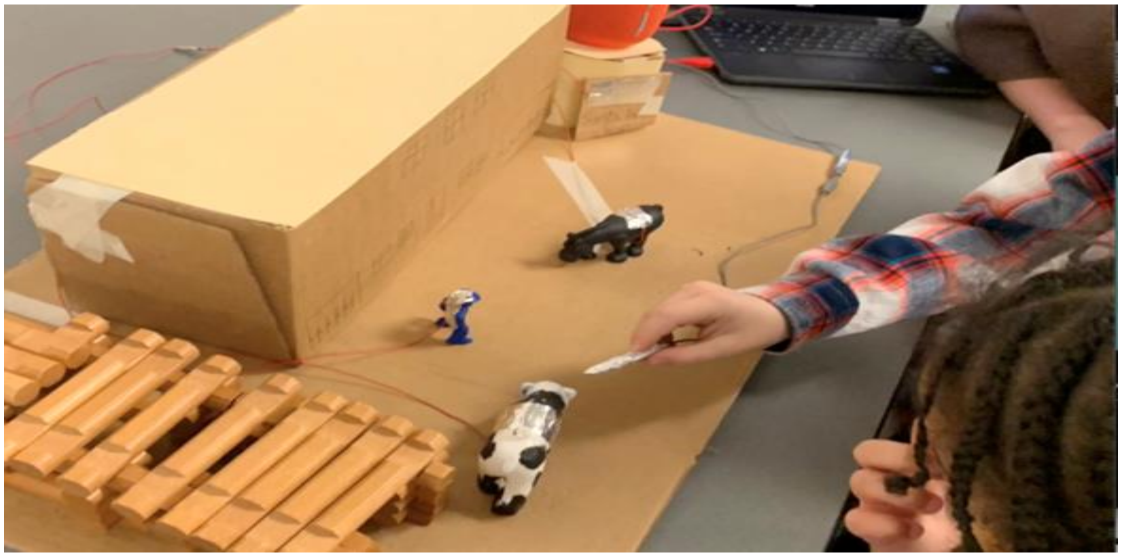


Figure 8. Picture of museum exhibit from S4's class.

Perceived value. S4 reported that programming was so popular during class time that she began giving students time to create more projects during elective time. Students began quitting choir in order to spend more time coding. A group of girls spent their recess and lunch time in S4’s room in order to create websites and programs to fundraise for dogs in shelters. S4 noted that coding in class created a culture of creativity and problem that continued beyond classroom projects and into the lives of the students. S4 described this cultural shift of problem-solving using technology as the 21st century mindset, which not only makes her students more marketable, but also empowers them to create in every aspect of their lives. S4 noted, “It's really preparing kids for a whole 21st century mindset. So to me now it's bigger than just coding itself. It's the whole computational thinking and that whole mindset of how can we solve problems using technology.”

Advice. When asked what advice she would give to others, S4 said to jump in and play with it and figure things out as you go. She warned that teachers are not going to be able to know everything and fix every problem; nor should they since students will figure things out with time and support. S4 advised teachers to shift their identity to be okay with learning with students. Finally, S4 provided an alternative for teachers who do not have technology or are afraid to use it. S4 said:

> But I would also say, don't be afraid to give it to your students without knowing everything. Most kids are very tech savvy-- that you just give them like a drop and give them time and they will wow you. Like they will just take off. And I think a lot of teachers are afraid to kind of unleash that.

She encouraged teachers to use unplugged activities in order to engage students in computational thinking and creative problem-solving.

S5. S5 has been teaching for over twenty-five years. Her undergraduate and Master's degrees are in special education, and she has a single-subject mathematics credential in the state of Pennsylvania. She has five years of experience with students in alternative schools, and twenty years of experience working with K-12 gifted students. S5 teaches in a small suburban district in the northeast of Pittsburgh. S5 considers herself techie in that she does techie things in order to provide more experiences for her students, meaning that she integrates technology whenever she can to further engage students.

Source and reasoning. S5 reported that she took a free development class at Carnegie Mellon about educational robotics, where she was introduced to Create Lab's array of educational robots, including the Hummingbird:

> When I got involved with the Create Lab at Carnegie Mellon Science Center. So I actually saw the hummingbird kit in the beta stage. I took a class called educational robotics for one credit at Carnegie Mellon. And my main motivation for signing up for

> that class and I paid for it out of my own pocket was to find something interesting for my students.

There, she noted that she was introduced to Scratch and placed on an email chain that provided ideas and new learning opportunities. S5 described a rich professional development culture in Pittsburgh that focuses on programming and STEM ideas. She noted that some of her younger students were also interested in Scratch, and so S5 began searching YouTube and online forums for ideas to motivate and engage her students.

Perceived challenges. When asked about challenges, S5 identified two major barriers to integrating programming in classrooms: student gender roles and teacher identities. She reported that girls tend to be more fearful of coding than boys. Despite the fact that girls tend to code better than boys, she notices that boys have more confidence, and often usurp the role of coder during group projects. Additionally, despite five support teachers at the school who can help elementary teachers use programming, she asserted that it is difficult to convince teachers to agree to take on coding projects. She said that teachers are afraid of the classroom chaos inherent in coding projects. Interestingly, S5 also noted that humanities teachers tend to resent STEM fields due to state pushes for more math and science focused state testing that often take resources away from the humanities:

> I feel like social studies and English like they're people like I don't know how to put it. It's sort of maybe a little resentment that their areas are not appreciated right now in this world if that makes sense. Cause like everything is STEM.

Although not as serious as student and teacher identity, S5 also acknowledged the fact that tech issues add to teachers' fear of integrating technology in their classes.

Instruction. S5 reported integrating coding and robotics into poetry and social studies lessons as a means of engagement and problem-solving, situated on content. S5 noted that she develops rubrics for features and content, and students have creativity in the technology they use as well as the product they design. S5 reported:

> They'll pull their Greek god from a hat and so then they'll have so much time to research. I have a rubric for research the Greek god they have to write a narrative. And then they have to make a robotic diorama for their Greek god. I mean it's high flying kids, I mean a lot of them can get some really cute stuff done in like three hours.

The rubrics require students to communicate their understanding of Greek gods, and specifically identify coding and robotics mechanisms such as creative use of looping structures and servo motors in their communication. Because the projects are large and require time and resources, S5 acknowledged that she only integrates coding into her classroom twice per year as a summative event of learning. Another example of a project from her class is when students recreate scenes from Romeo and Juliet using robotics kits. Figure 9 shows an example of a student project using LEDs, speakers, and servo motors for the balcony scene. For student support, she places students into heterogeneous groups. She reported that she does not expect all students to code; rather, one student will always step up to code while the others focus on *blinging up the project.*

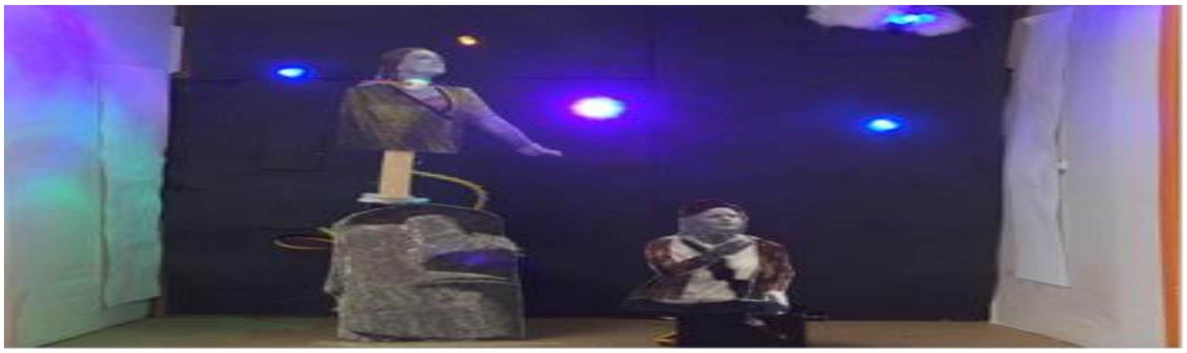


Figure 9. Shakespeare robots in S5's class.

Perceived value. Despite her observations that boys are more confident in coding and tend to take the coding position away from girls during group work, S5 described an increase in computer science interest at the middle and high school levels. S5 acknowledged:

> One thing I noticed you know somebody rises up and is willing to be their coder and then you know usually the groups are three to four in a group and then you find you know a lot of kids are either shying away from it or are afraid that kind of thing.

\She also observed that average students have more confidence in their coding when they reach the middle and high school levels; it’s not just the advanced kids who can code anymore.

Advice. S5 reported that a certain personality rather than younger age is required for teachers to integrate coding well. The personality she refers to requires an openness to try new things, acceptance of a level of chaos in the classroom, and the ability to learn with students. S5 noted:

> I guess until that teacher feels like OK you know because right now it's not just somebody being a sissy or being lazy you know. I think they’re just trying to protect themselves. I think there's a strong sense of responsibility to not conduct something that has some chaos to it.

She said that not knowing content is scary for many teachers, and urged teachers to try one or two things in their classroom per year, get comfortable with those technologies, and keep adding to it.

S6. S6 is a 12th year teacher who works at a private Catholic independent laboratory school in Pennsylvania. The school is preschool through eighth grade, and although S5 has always taught 7th through 12th English, she currently teaches at the elementary and middle school level after earning her Master's in K-12 language arts. S6 does not consider herself to be techie, but comfortable enough with technology to use it in the classroom. S6 reported that when she began incorporating programming into her language arts class, it was incredibly intimidating, but found that it was easy to do. In her words, "And so I still don't consider myself to be pretty techie. I wouldn't say that, but I feel comfortable enough with technology to teach it to my class."

Source and reasoning. S6 explained that as part of a larger STEM movement, her school started to push coding by offering a lot of trainings using the BirdBain Technologies robots from Carnie Mellon's Create Lab. She described one teacher in the school who serves as an expert user of the Hummingbird kits and convinced her to try using the robots in her class despite her trepidation. She reported to use online communities to allay her fears, as she found a teacher who uses the Hummingbird kits with poetry projects. She explained:

> We have a teacher at our school who's very much involved and tied in with the BirdBrain Technologies. And she actually kind of asked me if I would be interested in doing something in my class with the hummingbird kits. And at first I was a little like, don't know what I would do in my class with it. And then I started to do some research online.

> And I came across other people who were doing poetry projects using the Hummingbird kits.

S6 described her realization that she could give projects that focus on language arts, but still allow students to problem-solve and be creative in their products and thinking. She said that all students can get on board; there is something for everyone, which increases engagement for all students. To date, S6 reported to have been using the Hummingbirds in her class for three years.

Perceived challenges. S6 sited tech issues as a consistent classroom challenge, ranging from Wi-Fi, power issues, robots deleting downloaded code, and lost work. She explained, “Anything can go wrong. And the lesson we've had times when the power has gone out, you know, the Wi-Fi is down. Things like that. That's difficult.” Aside from lack of time collaborate with colleagues, she did not provide any other challenges to her activity.

Instruction. In a typical project, S6 reported to give students options for a poem or they can choose a poem they wrote. She said she urges students to choose a poem with rich visual or auditory imagery in order to make the product more effective. Students use the Snap! Language to use robots to create a visual and auditory representation of the poem. One example is a woman rising up as a visual aid for Maya Angelou’s *Still I Rise*, as shown in Figure 10. To start a coding lesson, S6 reported to give students step-by-step directions on each of the features of the robot using her own robot and computer as visual aid. She also explained that she prints out resources for students from the BirdBrain Technologies website, and provides links and resources from third party sites. S6 describes her class:

> With the hummingbird kits, they actually use Snap! programming. A lot of them are already kind of trained and Scratch and Snap! is very similar. So it's easy for them and the kids who aren't trained, they're a little overwhelmed at first, but then they realize how

easy it is. So I trained them in that part. They just have to do an analysis of the poem. So that's where the language arts comes in. And then they make the visual representation. A lot of it may look like a diorama kind of thing.

After the students create a product, they reflect on their experience as well as present their project to a group of parents. When asked about grading, S6 responded that although students are mostly graded on their creativity, she assesses students with a rubric that they receive ahead of time, with features that must be included in the project such as LEDs, vibration, and specific uses of the motor. S6 describes an instructional modification that she has made due to the coding project. Rather than looking like a traditional classroom, students are sitting everywhere in the room, clumped in groups, even on the floor. Students move around, and remain active throughout the project. S6 sees her job as facilitating, monitoring, and helping students when required.

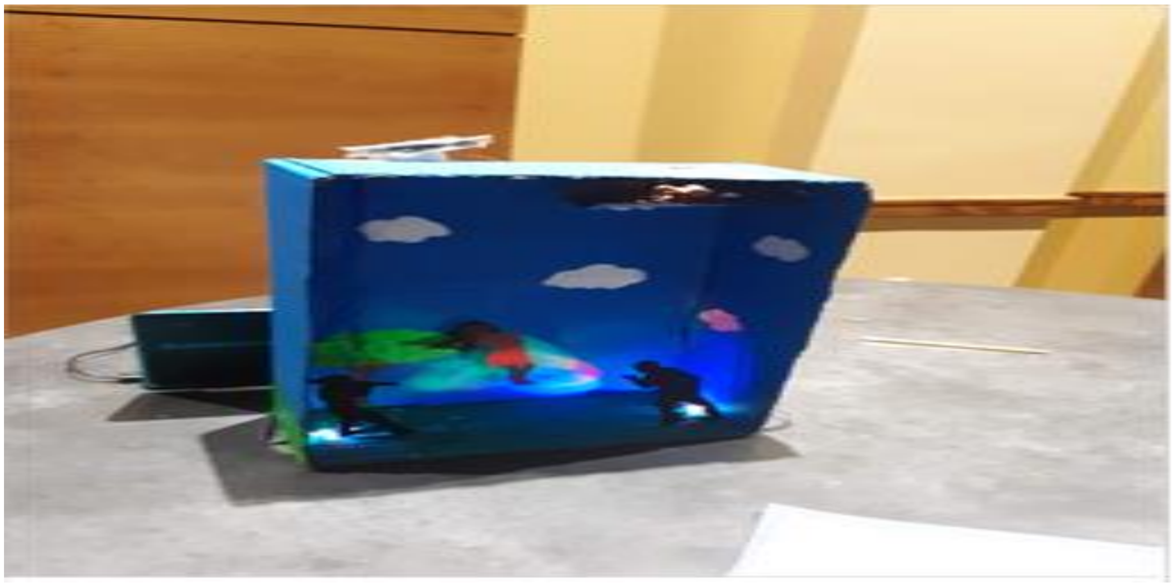


Figure 10. Robotic poetry in S6's class.

Perceived value. S6 reported that from the student reflections, she receives very positive feedback. Students who are overwhelmed at first and do not have much experience with programming say that they thought the project would be difficult, but really is not. The only negative feedback S6 reported to receive is regarding group work. Furthermore, S6 noted that during the coding project, students are better behaved than any other time during the year. S6 explained:

> I do this project with 7th grade, so they always say the most difficult part is agreeing with our group members. Do you know how they're going to do the visual representation of the poem agreeing to what exactly they're going to program into the board? But honestly, the actual coding part, I always get positive feedback.

Almost all students are engaged in the project, everyone takes a lead during the coding part.

Advice. When asked about what advice she would give others, S6 advised others to just try using coding. She reflected on her initial feelings and doubts as someone who is not techie and did not know anything about coding. Even after professional developments regarding the use of Hummingbirds in class, it was not until she actively tried it and understood how it functions in the classroom that she reportedly felt comfortable. In fact, S6 recommended that teachers embrace coding since the coding project is her favorite thing to do in the entire year. S6 explained:

> I would tell them if they had the reservations that I did in the beginning, like for the fact that I'm not one of those techie people and I absolutely knew nothing when I started. You know, we did the professional development and then I knew a little bit more, but I still don't feel like I could teach it. Once you start teaching it, you start to understand it more yourself.

S6 concluded that the end products are worth the labor, and students get a more impactful learning experience than just reading the poem.

S7. S7 teaches in an affluent suburb of Philadelphia. She has a Bachelor's degree in elementary education and a Master's degree in STEM education. She is going into her 20th year; eleven years as a second-grade teacher, six years as a fifth-grade teacher, and two years as a gifted support teacher. She currently works mostly with fourth, fifth, and sixth graders. Although S7 currently has a specific class for gifted students, she also provides push-in courses, where she inserts herself into another teacher's class for a unit or project. S7 considers herself very techie, particular in her environment, where she is the go-to person for tech challenges and support for tech integration.

Source and reasoning. S7 reported that about ten years ago, she was introduced to Scratch through her colleagues at a conference in New Hampshire about constructing modern knowledge. She recalled that at the conference, she befriended one of the original developers of Logo who would sit with her and talk about programming, and even brainstormed ideas for classroom. Consequently, S7 brought Scratch back to her second-grade students during computer lab time, where she said she learned along with her students. S7 explained that later, she began to integrate coding into her social studies projects. She sited her initial reasoning as wanting students to make something meaningful. She said that students do not learn to code for the sake of code; rather, they learn to code because they are trying to make their creativity and interests a reality. She noted that during a coding project, students figure things out, employ trial and error, dig deep into research and pull out the most important parts, and connect the research to code. S7 quoted:

> Well, I can think of two reasons. One is to make it something that's meaningful. So you're not just learning to code because I want you to learn how to code or you want to learn how to code… The second thing is the time that we have. So there isn't time for us to

> have, you know, computer science class. You know, there's barely time to do what we do. So now I'm doing a social studies project and you're using coding for social studies.

She reported that coding gives students the power to create what they are thinking, and S7 observed that they want to continue learning to code; often, students will sit in groups in her class outside of class time, creating new projects on their own. Unfortunately, S7 acknowledged the school does not offer computer science as a stand-alone course so S7 described herself as providing a necessary service for the students by integrating code into her social studies class. S7 said, “There isn't time for us to have computer science class. You know, there's barely time to do what we do. So now I'm doing a social studies project, and you're using coding for social studies.”

Perceived challenges. The only challenges S7 described in her classroom are due to student identity and behavior. She reported that students who do very well in school tend to struggle emotionally when their code does not work. Similarly, students on the whole are not willing to persevere and figure things out. S7 described Scratch as a media that requires clicking and playing and trying things, and lack of perseverance can halt a project. As the school moves toward STEAM learning and encouraging all teachers to use technology, S7 described some barriers for others. Primarily, the school wants teachers to develop STEM lessons, but S7 noted STEM as a mindset, a cultural shift, and not just a lesson, which gives teachers a false starting point. S7 reported that she is afraid that the school’s “every student has the same experience” mandate will impact her practice. S7 said:

> I want everyone to have the same experiences. I'm not sure how that's going to look next year and the years going forward, because especially timing. You need to train the teachers. You need time for them to include this in our heavy loads.

S7 acknowledged that teachers require training; many teachers in the school are willing to try new things and administration promotes innovation, but there is a different fear about using code than for example using a generic app online. S7 said, however, that organizations like Code.org are changing that by making programming easier for teachers.

Instruction. S7 reported to give students very open-ended projects, where coding is a tool that engages and situates learning. For example, her fourth, fifth, and sixth grade classes do a project called Why Do Humans Make Things? She describes students performing in-depth research on the content of the course (Native Americans, early colonies in the United States, ancient civilizations), and demonstrating their learning in any way they want as long as they use code. For example, she portrayed a group of fourth grade students creating animations in Scratch. To support students, S7 reportedly demonstrates introductory skills related to the language or technology students use, and students are given a short activity where they practice that skill such as animating their name. S7 explained that as students begin their projects, her instruction becomes more individualized to the needs of the group as she supports whatever features students want. She declared that often, she will use other students as teachers during the individual support phase. For example, she explained that one group wanted a train to look like it was moving so they wanted the background to scroll. S7 noted that she could not figure it out, but asked other groups if they could, and they did. In her words:

> She was trying to make a scrolling background to make it look like the train was going. And so then kids in the other classes would sit and try to figure it out, and maybe they got stuck. So we were all trying to figure it out. And finally we did. I had one student who would make it. He's like, I figured it out. I'm going to do a tutorial. He put notes in it, shared it with us. That was pretty awesome.

S7 reported that students are assessed on creativity, not code.

Perceived value. S7 reported that students who traditionally do well in school struggle with coding because they want to be correct immediately, and struggle to preserve; whereas students who typically struggle with reading, writing, and math tend to love Scratch due to the ability to play around, try things, and explore different avenues. When asked about student impact, S7 responded:

> The interesting thing that I found from that experience and from with that group of kids, as I did it for several weeks, the kids who struggled the most in school, you know, they struggle with math, reading. They seem to be most at ease in this environment of the unknown, where they just click things, figured it out. And kids who were very comfortable usually in the classroom, really had a hard time plowing through it.

S7 also acknowledged that boys seem to be more interested in some of the projects than girls. Moving forward, S7 described her plans to find new connections between coding and mathematics in order to benefit all students.

Advice. S7 recommended that teachers try to integrate coding without having to be an expert. Despite her willingness to work with other teachers, she reported to often ask multiple times if she can come in and work with teachers on delivering a coding lesson. For teachers who are afraid to teach something they do not know, S7 advised to use peer pedagogy; to allow peers to support each other:

> One thing that I suggest encourage is to try to be okay not always necessarily being an expert, but being able to introduce this to students. And I think an offering to help [students] is, you know, another thing. I'll do it with you, playing with you.

If a third-grade student knows much more than a high school student, she reportedly would group those two students in a heartbeat because it is so empowering for both students to learn from each other.

S8. S8 was a technologist for software engineering tool companies before becoming a teacher. He has taught middle school science for thirteen years in southern California in a small private school where he is the only science teacher in the school. S8 has a Master’s degree in educational technology, and considers himself very techie due to his background.

Source and reasoning. S8 reported to begin using code in his science class because code as a way to express logic, and provide his students with an entry to systems of cause and effect. S8 described his entry as using Microsoft Excel to run formulas, but after an introduction to other technologies during his Master’s classes, he continued to use Scratch, Makey Makey, and NXT Lego Mindstorm kits for their user-friendly interfaces. S8 said:

> My objective was the concepts, which is why I use a GUI, graphical user interface, like that or like the NXT language for the robots for my purposes is really good. Definitely as a tool, not as something as an end on its own, but following the idea of identity formation.

S8 described his attention to his students’ sense of identity and focus on creating classroom experiences and structures that enforces students’ sense of self as scientist. He portrayed coding as a means of easily creating dynamic lessons, as students cycle through the scientific method as they play with code. S8 reported that he wants to get as far away from old white guys holding Erlenmeyer flasks as possible, and if Scratch can be used as a storytelling device for communicating ideas, then he will try it. In fact, S8 said, “And that's pedagogically known to be more effective as far as getting away from the classical… old white male Einstein with an

Erlenmeyer flask in his hand to the kid visualizing himself." Ultimately, he described coding as a tool for deeper and more engaging learning.

Perceived challenges. S8 observed that creating an interactive lesson for delivering spreadsheet or coding procedures is difficult and are never executed well. He reported that once he gets to a project, students are able to be creative and apply their knowledge of science in interesting ways, but getting there is difficult, laborious, and S8 recounts as almost not worth it. S8 described a class where teaching the technological skills did not work well:

> If the example isn't prepared pretty carefully where there's just enough scaffolding in place that I can then say, now, everybody, what you want to do is click on this cell… And now type in so and so whatever. So again, the two dimensions are how good is the preparation—how effective is the scaffold—and the character of the kids, the character of the class.

He discussed his role as an older white guy teaching computer science to diverse students, and explained that the progressive identity of his school benefits his practice in that the culture of the school emphasizes breaking traditional barriers in access to segregated fields. S8 reported that two thirds of the students come from families who work in the Hollywood industry, and students question traditional systems of power. For example, S8 remarked that you just do not hear "That's so gay" anywhere on campus. He concluded that increasing women and the limited number of students of color at the school, then, has not been difficult.

Instruction. S8 reported to use coding in multiple ways: as an engaging tool for use during the science fair; as a medium for students to tell narratives and communicate ideas in Scratch; and as a means of making scientific calculations and processes easier. S8 noted that each of these intentions require different implementations and considerations for coding. For example, S8 described the science fair as requiring open-ended and individual support, as everyone is doing something different, and the variation in technology can be large. He reported that telling a story in Scratch requires basic understanding of a few blocks in the language followed by time for students to create and communicate. Finally, he mentioned that using code as a tool to perform science requires more structure, stronger rubrics, and a more traditional classroom. For example, when discussing how his students use formulas and programming in Google Spreadsheets to calculate the nutritional facts of their energy bar recipes, he described how he provides specific coding knowledge and a detailed rubric.

Perceived value. S8 reported that coding lessons and projects increase engagement significantly. S8 described his focus on urging students to build and make in order to learn tools that could serve them in the future. S8 said, "Whether it be a spreadsheet, whether it be scratch or whatever, when a kid gets it and you get the aha, and the kid realizes I now have a tool in my tool kit." S8 reported that since he started using coding in his class, he has noticed that the "big Minecraft kid" who "builds Linux servers" has disappeared from his class, which he doesn't think is necessarily a bad thing. In fact, S8 purported to be aware of the need for more diversity in computer science, and whether due to the culture of the school or providing access to computer science, he noted that all students have access at his school and are reaping the benefits from learning a new tool.

Advice. When asked about what advice he has for other teachers, S8 urged teachers to open themselves to accepting the fact that students might know more than the teacher with respect to coding and coding environments. S8 reported that teachers require training, and in order to train teachers to use code and technology in general, S8 advised using hands-on non-flashy methods of showing teachers how to use it in practice, using the same inquiry and equitable methods that a teacher might use when presenting it to their students. S8 mentioned, “Are there ways you can deal with the anxiety? Yes. Not to convey information, but give them an experience which overcomes the activation energy, the emotional hump, preventing them from getting involved.” According to S8, allowing teachers to empathize and feel like students learning a lesson will give them practical tools that they can use.

The Recontextualizing Phenomenon

Recontextualizing code is a phenomenon, and this study served to help to understand that phenomenon through the lived experiences and perceptions of those engaged in it. All eight of the informants in this study were intentional in their use of coding in the classroom. Although the initial reasoning and perceived value of integrating code in the classroom differed among informants, their initial objective was not to integrate code; rather, to modify students’ experiences. Coding then became a catalyst for change in the classroom in order to improve student engagement and experiences.

All of the informants reported to use coding as a tool for enhancing students’ experiences; that is, the informants took the initiative to create a dynamic change in their classroom. Every informant reported to be supported in their endeavor to learn coding as well as in their implementation of coding in the classroom. Teachers learned basics of coding and best practices in implementing coding in the classroom from former participation in industry, local workshops, on-site colleagues, and online communities. Interestingly, each of the informants chose one of two programming languages: Scratch and Python. Scratch was used for interactive experiences and narratives while Python was used for calculations and simulations. Scratch and Python were both designed for learners, and to be less complex than industry-standard languages.

All of the informants who teach in the humanities were alike in how they implemented code: programming languages and programming environments became encapsulating media for students to communicate their learning and express their individuality. This included students building narratives of their learning in Scratch, interactive museum exhibits, and dynamic dioramas with robotics. Teachers in STEM fields integrated code as part of their standards to mediate students' learning. This included building short code in Python to execute trivial calculations in a Geometry class, building a simulation in Microsoft Excel to test different variables in a system, and using Scratch to build a simulation of density. Despite differences in how humanities and STEM teachers integrated code in the classroom, all informants perceived coding as an activity that all future citizens should know, and introduced coding in the classroom in order to increase coding literacy. Whereas coding used to be seen as an esoteric skill, relegated to those who will one day become computer scientists, the eight teachers reported to expand what it means to do computer science by interpreting coding as a digital literacy.

Each of the eight informants described his/her identity as a teacher shifting from possessing all the knowledge to a more knowledgeable other engaged in learning with the students. The types of learning experiences that teachers described were constructionist, which aligns with facilitating learning rather than transferring knowledge. Additionally, all informants described a positive impact on students. Some teachers perceived coding to increase participation, engagement, and attitude in class. Others perceived coding to form new identities for students.

To summarize, the eight informants described their lived experiences as participants in the phenomenon of recontextualizing coding. Each of the eight informants was a self-advocate who intentionally recontextualized programming as a means of increasing coding literacy as well as altering students' experiences in preparation for 21st century careers. Each of the eight informants was trained and supported by his/her respective community, and interestingly, chose similar programming languages and environments specifically designed for learners. Each of the informants described a process of trial and error as he/she gained experience in utilizing code in the classroom. All eight informants described their identity shifting from a *sage on the stage* to a more knowledgeable other engaged in the learning process with students, and perceived a positive impact on student behavior, identity, and engagement. The shared experience among the participants engaged in the phenomenon of recontextualizing coding involved self-advocacy, interpreting coding as a digital literacy for 21st century activity, significant training and support by local and online communities, the predication to iteratively and incrementally add new technologies and code, a shift toward a more constructionist pedagogy where the teacher learns with students, and the perception of benefiting students both in the classroom and long term.

Expanding What It Means to Do Computer Science

The eight subjects each describe pedagogies, activities, and reasons to integrate programming into their respective content domains. Content coded by *activity example, assessing code, impact on students, instructional modification, programming language, support student code,* and *teacher style* were collated by theme and organized by content with the purpose of discovering common experiences and perceptions among participants related to using programming as a teaching vehicle. While expanding the activity of computer science, informants described: programing languages and environments that mediated students' activity; and reported to use coding as a tool for student engagement or to moderate understanding of content.

Programming Languages and Environments

Informants reported using programming languages and environments with a low floor and high ceiling, designed specifically for use in the classroom. S1 described Scratch as being highly visual and accessible to all of her students. During the interview, she said,

> The diversity you can use in those languages to do more things like graphics and things that you cannot do in Excel… Using, even scratch you can see it… I had my students draw a chunk of ice and see how long it will take to melt, and they have beautiful sprites that change like, shrinking the ice in the right amount of time… it's more visual.

S1 described the ability for students to easily animate while modeling real-world phenomena makes Scratch an ideal environment in her class. S2 reported to use *Scratch* and *Thunkable* in her class because students "do it for fun and they show [her] all the random little apps they build." S2 described Scratch as a programming environment so engaging that students spend free time creating side projects. S1 and S2 described using programming languages that are designed for learners and easy to use in order to increase engagement.

S3 reported that Python fit the needs of his co-taught special education math class. In his words, his goal was

> to make them feel successful with some very easy to use code… I would not want to give C or Java as the first programming language. If that's your first formal course, you're going to like go crazy because there's so many things you have to learn with brackets and headers and IO and this and that and all this stuff to do things. Python is more abstract. Just type in X equals Y and Y equals this and multiply and it's code that's a very—there's a lower floor.

In this case, S3 reported a distinct difference between lower-level programming languages such as Java and Python and higher-level languages like Python, and bridged student level with the complexity of the programming language.

In addition to being a tool for engagement, subjects reported Scratch as a tool for constructing artifacts. During her interview, S4 reported, “when scratch came out and things that made it accessible for kids to start learning… they're not learning syntax, but just be motivated by creating something that works and does something.” Similarly, S5 reported on two versions of Alice, and noted that ease of use and time constraints determined her choice: “I moved from the storytelling version of Alice which was very easy. If you want somebody to walk you know you could just walk; whereas with regular Alice you have to do every little bit of animation yourself.” Alice is a three-dimensional object-oriented language designed for learners. S6 also reported that the ease of use of Scratch and Snap make them ideal languages for elementary school students, and added that because many learning technologies interface with Scratch and Snap, they can be used across multiple systems. S6 said that her students:

> Use the Snap programming... Apparently that's what runs best on Chromebook. And with the hummingbird kits, they actually use Snap programming. A lot of them are already kind of trained and Scratch and Snap are very similar. So it's easy for them and the kids who aren't trained, they're a little overwhelmed at first, but then they realize how easy it is.

S7 also described a connection between block-based programming languages and use for robotics and other learning tools:

> I really have only used block-based type programming with the kids. I find most of the kids are happy with and there's a lot that they can learn just with the block based. So that's what we've used for, like, for the Finch robot.

S8 also reported that he uses block-based languages like Scratch due to their cost-effectiveness and because he had learned about it in his previous schooling. S8 mentioned,

> now I see either Scratch or how the other Makey Makey interface works more. What the heck. Let's let's fire up Scratch. I'd read about Scratch. Used it. Probably used it in some of my Master's classes in computer technology, you know, classes that I attended as an adult and thought, hey, you know, I got to put this to work, which was pretty easy to do. It's certainly cheap.

In order to construct across a large set of technologies, some teachers reported to use programming languages designed for learners that interface with many different systems in order to reduce complexity and use what students already know.

Interestingly, the programming languages and environments that subjects reported to use align with the research in that teachers reported to choose languages and environments designed for learning rather than industry-standard languages. Using the design philosophy that programming languages should have a low floor for access and a high ceiling for future opportunities of complexity, the Scratch language uses visual blocks and a community of users to support each other (Resnick, Silverman et al., 2009). Visual blocks and large libraries of code such as contained in Scratch reduces the distance between the user and the computer (Anderson et al., 1988), and uses abstractions to modify the interface of the programming environment to more closely reflect human approaches to problem solving (Smith et al., 1996).

Coding as a Tool

Eight out of the eight participants referred to coding as an instructional tool. S1 described computers and coding as a tool for learning for science, similar to how a notebook is used by students in class. S1 reported, “They are using the computers as a tool in order to solve a problem… It is like a notebook.” While describing an instructional change in her classroom, S2 reported that “there's a lot that you could do on [Scratch] that was similar to Google Slides. Be able to put in multiple little scenes and stuff and I really like that.” S2’s account of using Scratch likens the programming environment to presentation software, a tool for students to communicate their learning. S3 reported programming to be a tool that allows students to check their work and calculate: “I want to have them look… it's just a computer, just a tool that they allow them to create… a little computer program could help me do the work a lot more accurately and a lot more efficiently.” While listing one of the many computational artifacts S4 uses in her classroom, she reported, “just by giving them a little bit of foundation using Google CS First, it becomes a tool and not every kid in my class uses it, but it is a tool available for all of my students.” S5 reported to use programming in her gifted classes as a means of giving students a tool for engagement and to heighten interest. When asked about what prompted her to use coding, S5 replied, “I actually saw the hummingbird kit in the beta stage. And my main motivation for signing up for that class and I paid for it out of my own pocket was to find something interesting for my students.” Similarly, S6 reported that coding is “one more tool that students can use in order to engage in content.” While describing the development of computational artifacts designed for classrooms, S7 reported, “And now there are tools that we have—such as the Finch and Hummingbird—and I have Microbits, things that I can use with the kids.” The Finch and Hummingbird robots as well as Microsoft’s Microbits are programmable artifacts designed for classroom use. Finally, S8 reported that “something like Scratch gives you graphical primitives, which are which do something as a basis, as a as a function of whatever variables are doing and whatever an input such as the Makey Makey is doing.” Here, S8 describes programming as a tool for delivering science content, specifically cause and effect using primitives and variables.

The focus on coding as a tool is interesting to note. The initial value of learning programming was a means of mediating mathematical knowledge (Feurzeig et al., 1969; Harel & Papert, 1990; Papert, 1972; Solomon & Papert, 1976). The object of programming changed from a mediational use for mathematics to a subject and content area of its own when computer science became a subject of its own within schools (Aspray, 2016). As such, interventions to increase diversity in computer science have been considered under the assumption that computer science is an entity of its own (Fields et al., 2015, 2016, 2017a; Goode et al., 2014, 2018; Kafai, Fields, & Searle, 2014b; Margolis et al., 2015; Perković et al., 2010; Rennert-May et al., 2012; Resnick et al., 2009; Settle et al., 2013). From the reporting of the teachers in this study, some teachers now consider coding to be a tool for engagement and problem solving. As such, coding could be included as a tool within educational technology.

Teacher Activity: Curriculum and Instruction

Since every subject reported to perceive coding as a tool for use in the classroom, it is useful to investigate the way in which they utilized code, looking at content, the environment, their perceptions of the value of coding, how they perceive coding to impact students, and how they assess student code.

While describing their instructional practices, the value of coding, and the perceived impact on students, narratives of how content and coding began to emerge in the data. Two dominant themes emerged regarding how teachers reported to use coding in the classroom: coding as a means of communicating ideas; and coding as a means of mediating content standards. For each of these themes, subjects described instructional practices, perceived value of coding, and perceived impact on students will be presented. Artifacts of lessons and student work will also be used to support the claims of the participants.

Coding as an Encapsulating Environment for Problem Solving and Creativity

All of the humanities teachers and one of the three science teachers described using code and coding environments as a means of engaging students in problem solving and creativity while also working within the content standards of their domain. The following presents subjects who reported to use coding as a means of communicating learning.

S2 and instruction. As an agricultural science teacher, S2 reports that she uses code to create engaging environments for student learning. While describing her initial reasoning for using code in the classroom, S2 reported,

> I want the kids to be creating their own to show their learning through their own creations and not specific ideas. Sometimes it is with a specific app or smash two together. But just to be outside of the, just essays… We can make movies; we can do far more things. We can code.

S2 described her use of code as a means of creating a structured environment in which students can creatively communicate their learning. S2 reported her awareness that she does not use code to teach the standards; rather to provide a space where students can be creative in their communication of the standards. While describing how she implements code in the class, she said, “everything builds, but it's kind of out of content.” For example, S2 reported that while teaching cellular respiration, her students make cookies with yeast and different amounts of sugar, and then build a website and app in order to market their business and increase their market value. Although the coding does not directly relate to cellular respiration, Scratch as a coding environment provides a structure for students to engage in content and communicate their learning while being creative.

S2 and perceived value of coding. S2 teaches agricultural science in a valley in California where the dominant industry is plant science. She began integrating code into her classroom as a means of exposing her students to the future of agriculture. She reported:

> They wanted my students with an end goal of becoming like mid-level management in plant science or ag mechanics or teach them to weld, which is all fine. Nothing against that. But really seeing when we went out to some of the farms and how computer science is being used. They had sensors to measure the amount of water that is evaporating and that was the middle of the drought. And it's like, no, there's stuff here, like these kids can go a lot farther than just learning to weld, you know. And so I just started. To me that was just the start of it, seeing like we have to do more of this.

S2 describes the low bar that others were setting for her students, and more importantly, the goals that were set were not realistic compared to the activities within the agricultural field. She therefore perceives coding as a tool that can support her students’ futures, knowing that most students go into agriculture in this particular region of California. She finds a way to make coding meaningful for her students, as they can use it in their future. S2 continued:

> And then I realized we as a state are far behind in education, but especially in the valley, you know, I heard in the Bay Area they have, you know, coding clubs and drone clubs and elementary schools and that their AP computer science, they have lotteries to get in. Whereas here we weren't even offering those classes at our schools. We have

> agriculture… And so, then it became more of 'this is important.' Even if the kids don't go that route, they should at least be exposed to it. It's an equity issue.

Not only does S2 contextualize coding within agricultural science to improve the futures of her students, but she perceives this act as an equity issue due to the access that others in more affluent areas of the state have compared to her community. She then provides access to computer science in order to give her students the same opportunities as others. Her recontextualization of code, therefore, is about identify formation and providing access to others, situated on the science and industry that is prevalent within the community.

S2 and perceived impact on students. S2 reported to provide classroom activities where students make meaning of the coding, situated on the dominant industry in their community. To this end, she described projects that use code as a vehicle for communicating learning and synthesizing information. When asked how her lessons impact students, S2 reported:

> They personalize them. They are going to remember them far more because they're going to be spending a lot of time looking at oh am I showing how carbon dioxide is coming from this cell. You know, they're really focusing on that because they're trying to do the programming and they know the content that they have to do… they're really focusing on their content. How to summarize it, because you don't want to see a huge, boring paragraph, you know, a paragraph to read. I think they just they'll remember it more because the kids—it was photosynthesis, they're focusing, we're going to have the sun, how am I gonna make the sun spin? And they want to make sure it's facing the plant and they draw their little arrows. So I think they're just internalizing it more.

S2 depicted coding as a mechanism for engaging students at a deeper level in the content. Not only do students need to understand the process of photosynthesis, but they need to code an animation in Scratch for the sun and the plants, and to represent the process visually. S2 argued that this deeper delve in the content allows students to internalize the material, and specifically identified personalization as a catalyst for engagement, which supports the findings of Kafai et al. (2014b).

S2 summary. S2's reporting implies that she perceives coding as a technological literacy that all future citizens should know (The National Academies Press, 2002). By using code as a vehicle for communicating learning, code transcends its previous cultural milieu as an esoteric language of technological development, and assumes a softer role as a method of communication and human expression. In an effort to provide access to computer science, S2 humanizes code for her students, allowing her students to use code for self-expression and as an encapsulating environment in which to play with content.

S5 and instruction. S5 uses coding similarly to S2 in that she reported that her students code robots and LEDs in circuits in order to communicate their understanding of a topic as well as synthesize their learning through symbolism. When asked how she uses code in the classroom, S5 gave an example of a project she does with seventh grade students:

> It was the novel for The Giver. And I happened to find an artist that does really cool art for The Giver. So I ran a bunch of that on paper and you know one day I went in there and I taught them how to do one circuit and then I showed them about parallel and their final project was to light up the art. I think I had like fifteen different choices for art. And then they worked with a partner and they had to light it up and instructed to be symbolic. You know if you pick a blue light I want to know the reasons why you pick a blue light, you pick a red light I want to know the reasons and you have you know tell me why you

> pick the color you did. So we lit up all this art related to The Giver and you had to talk about the color of your own and why they chose it and how it relates to the novel.

In S5's reported lesson, students do not create a narrative of their learning, as S2 stated. Students create art using programmable circuits, and their communication of their learning is embedded within the symbolism of their art. S5 mentioned that students must also submit an analysis of their project, stating how their art reflects their understanding of the material.

S5 and perceived value of coding. S5 reported to perceive integrating code into social studies and English language arts classes as an engaging and creative way for students to express themselves. In her words:

> I thought it was like a nice way to create a project that was a little different than say a physical world project. And then the opportunity to be creative—it provides an opportunity for kids to be creative… We took Greek gods and combine them with the Hummingbird and make essentially like robotic dioramas that low level kids that are usually unengaged, you know they were just into the project. Maybe they weren't into the social study learning about the Greek gods but at least they were engaged with the project.

Aligning with the use of code as an engaging method of communication, S5's perceived value of coding is embedded in the ability for students to be creative and to problem solve as well as increased engagement. Code, then, is being utilized like any other educational technology designed to increase engagement and support creativity. Like S2, code is humanized, and moreover, assumed as a tool for technological literacy.

S5 and perceived impact on students. S5 perceived the exposure to coding to increase the number of students taking computer science at the middle and high school levels as well as students' capabilities. When asked how coding has impacted students, S5 responded, "So what I'm seeing as we get it get up when kids start hitting the middle school. They have a better background and I kind of attribute it to what was going on in those two classes." S5 reported that access to coding, despite not always having computer science classes at the elementary level, increased participation in the middle and high school levels as well as increased the level of student knowledge.

S5 summary. Like S2, S5 reported to implement code as a digital literacy that all future citizens should know. She recounted that creativity and engagement are the values of coding, which align with media creation as a form of digital literacy. Interestingly, S2 and S5 are similar in their implementation of code and perceived value of coding. If one were to consider coding a digital literacy, then code would not necessarily need to directly mediate students' understanding of content; rather, code would be used as a vessel for creating and structuring activity.

S6 and instruction. Like S2 and S5, S6 reported to use coding as an encapsulating structure for communicating ideas and learning. She reported that she uses the Snap programming language and the functionality of Hummingbird robots from BirdBrain Technologies to structure and situate students' learning:

> I give them a list of poems to choose from. Or they can look on their own if they would like… So be something that would be, you know, very easy for them to make a visual representation of… And then they use the Snap programming to do this… And with the Hummingbird kits, they actually use Snap programming. A lot of them are already kind of trained and Scratch and Snap are very similar…They just have to do an analysis of the poem. So that's where the language arts comes in. And then they make the visual representation. A lot of it may look like a diorama kind of thing.

Similar to S5, S6 reported that students analyze a poem, and create a symbolic visualization of their analysis using robotics and coding as a means of communicating their learning. S6 emphasized that the language arts content is embedded within the analysis of the poem, and is further explored and deconstructed by distributing their symbolic analysis across the functionality and interface of a robot. Not only is coding used as a vehicle for communication, but also it mediates and situates the learning.

S6 and perceived value of coding. When asked what value she sees in coding, S6 responded that coding enables students to work on hands-on projects that include problem solving, critical thinking, and creativity while providing something for everyone:

> I think right now our hands-on projects got things that involve problem solving and critical thinking and creativity. And honestly, that has all of it. And I could still find ways to tie it in with language arts because, you know, they make their robot about a poem. They still have to analyze the poem. They have to make a visual representation of that poem. So I just felt like that all kids could kind of get onboard with this. You know, this isn't just something that just certain kids with just one particular talent, you know, would get involved with. It's something for everyone. And that's how I felt when I first looked into the project. And it turns out I was actually right. And the kids love this project.

Like S2 and S5, S6 reported to value the engagement, problem-solving, and creativity that coding affords students in designing some artifact. S6 adds to their recounting; she mentioned that integrating coding in language arts increasing engagement by broadening the interest scope of the project. Students interested in the language arts or STEM can participate; there is something for everyone.

S6 and perceived impact on students. S6's reported that focusing on broadening participation on her project worked. In her words:

> That was the best behaved they were throughout the entire year. It's about a five-week project and that was the most on task they have ever been. They honestly take pride in the project and in the product that they are putting out there. They know ahead of time that their parents are coming in. I'm inviting other teachers and students to see this. They know that everybody is going to see this and they want it to look good. And they honestly, you know, they want to take pride in it. I had students who they say language arts isn't really their thing and some of those kids honestly put the best projects together.

S6 noted that students take ownership of their projects, and have pride in their work. She reported to include community members like parents and other students to take part in the activity. Additionally, S6 recounted that students who traditionally are not interested in language arts engage and create unique projects.

S6 summary. S6's focus on increasing participation in her class using coding as a vehicle for engagement and communication was reported to be a success. This reported phenomenon aligns with previous research that integrating coding into non-computational classes can broaden participation among underrepresented youth in computer science (Eglash et al., 2006; Form & Lewitter, 2011; C.-C. Lin et al., 2009; Wolz et al., 2010).

S7 and instruction. Like S2, S5, and S6, S7 reported to encapsulate learning within a larger project that involves coding. S7 described her students building animations and games to communicate their understanding of the material in a social studies class:

> It's called Why Do Humans Make Things? So it's social studies… They had to do some in-depth research on a particular topic related to their social studies. We had categories

> and connect to that idea of why did humans make things so the fourth grade—they work on our local Native American history. And they learn about each state. This is the kind of thing where the programming is a tool… So, for example, there was a big group of fourth graders who wanted to learn animation. So they used Scratch and they just created animations. I mean, they drew the characters, they created the conversation. And that was to teach about whatever their topic was after they did the research. Sometimes it was a game that helped you learn about something.

S7 portrayed her use of coding as an educational tool like PowerPoint or a poster; that is, students use coding as a mechanism for communicating their learning of the content in the course. She explained that in Scratch, students make animations or games for others like parents and other students in the school to experience their vision and learning in an interactive way. S7, then uses code as a means of engagement with the added benefit of being interesting enough to make learning a community endeavor.

S7 and perceived value of coding. S7 reported that coding projects afford students the ability to deep-dive in the content and spend time problem-solving while also processing how to communicate the content, giving students a deeper understanding and appreciation for the content. She also reported that coding is interesting enough to unite the community:

> So whether that's the logic of figuring out what makes things work or, you know, the trial and error, really digging deep into their research while they're doing it. So pulling out the most important parts that they want to have connected to whatever they're programming. And then when they present, getting the interest of the other students and adults, really, anybody who had a game created when people would come in and look like a museum.

S7 depicted coding as a creative endeavor that all students can use to communicate and express themselves. Again, a subject reconfirms the emerging theme that coding is considered a digital literacy that can be used for creative construction and expression.

S7 and perceived impact on students. Earlier, it was discussed that S7 perceived her traditionally stronger students in school to struggle with not arriving at a solution immediately while other students who typically struggle in school feel supported by the trial and error method of writing code. Considering that S7 reported to use coding as a digital skill for communication and expression, she described how minority and struggling students respond:

> I've had, you know, some of our kids who are minority kids who are part of this and are also sometimes struggling students… and they're showing some good growth regardless of how they would typically perform in the classroom… Getting them involved and realizing that this might be a way that they learn and it might be easier for them, even as they're sitting in and working with others to learn, you know, it's not like they have to sit by themselves and figure it out that they are collaborating. And that might be something that's good.

Here S7 made a connection between traditionally underrepresented students in computer science, collaborative learning pedagogies, and access. Although S7 was not explicit in making this connection, it supports previous findings that relate collaborative pedagogies to increased engagement among underrepresented youth (Goode, 2007b; Margolis et al., 2014).

S7 summary. Much like S2, S5, and S6, S7 integrates code into her elementary level social studies class as a method of communication and self-expression. Students learn content and then create a project in Scratch in the form of an animation or game to communicate their learning of the content. S7 reported that traditionally academic students tend to feel frustrated to not get an immediate solution while struggling students find the trial and error aspect of coding beneficial to their experience. Although not explicit, S7 described her experience of building a connection between underrepresented youth, collaborative pedagogies, and access.

Coding as a Direct Mediation of Content Standards

STEM teachers like S1 (science), S3 (math and science) and S8 (science) did not report using coding the same as the humanities teachers. The STEM teachers reported that rather than using code to create an environment of problem solving, for them, code directly relates to their content standards.

S1 and instruction. When asked how coding relates to her content standards in science, S1 replied:

> They are using the computers as a tool in order to solve a problem… Right now it is by solving a problem or how to. For example, I'd give them like a rock and I say okay I have a rock and I need the density of this rock. How are you gonna solve it. I'm a robot and you need to program me to get the density of these so give me simple steps and easy to understand simple and accurate steps.

In this example, S1 described the use of code to simplify and solidify students' thinking about the science concepts. Interestingly, S1 said, "For using computational thinking—when I need to review. Especially with these kids that they need to be seeing things more as simplified." Here, S1 describes computational thinking as a means of simplifying content, or paring down content to its basic ideas in such a way that a computer can understand.

S1 and perceived value of coding. In addition to simplifying problems, S2 reported to perceive coding and computational thinking as a means of making the students' processes visible. While describing how she focuses on the process over the product, S1 said:

> I need to see where you get that answer. How do you enter this equation? How you move this letter here and there. So I teach them to write that process in the notebook or in the board to see where they are. And in computer science something like that you cannot memorize that code you need to write it down. You need to see the steps.

It is implied that the focus of algorithms and deconstructing problems in coding and computational thinking creates a structure for students to make their process visible. In this way, S1 perceives computer science to support the acquisition and understanding of science concepts by deconstructing problems and identifying step-by-step processes for problem solving.

S1 and perceived impact on students. S1 reported a contradiction in how her students respond to coding. When asked how students respond to her computer lessons, she said, “Well the thing is when I work with computers, they don't want to exit my class. They want to stay longer.” Although S1 described students as wanting to stay in class to work on programming projects, she also reported that some students did not react well to coding. When asked what challenges she faces while integrating coding, S1 reported:

> My science class some kids didn't understand… why we need to do this. This is just a computer. Yes. But I need to see how this works. And you can show me a screen because some kids are, they had different styles of learning. And they don't want to do it that way… those that have different abilities—they refuse to do it. They struggle with a computer. So when you say Computer Science For All I'm seeing kids throwing the keyboard—those kids who have like emotional disturbances. They cannot handle frustration.

S1’s reporting demonstrates conflict in her beliefs about computer science education. Although she uses coding as a means of making thinking and process visible in her physics class, she grapples with the idea of “Computer Science for All,” a popularized slogan that gained traction from NSF-funded grants and projects designed to increase participation in computer science education at the K-12 level. Interestingly, earlier in the interview, S1 provided information that could explain the frustration of her students:

> [Administrators] are more looking for the STAR test numbers and how these tests affect the students. Even in science, I realize that at the end of the year my student didn't learn nothing at all from science. They know how to answer the tests. They know how to word the steps quickly and strategies to answer the tests. But they don't know science at all. They don't know the concepts. They know, like machines, how to answer this question but they don't get it. They don't give the information—they don't have passion for science, and at the end we lose these kids.

Like S7, S1 makes a connection between testing culture and student frustration at not getting to a solution immediately while coding. This tension between testing culture and integrating code in a physics classroom manifests itself in the way in which S1 reported to perceive student impact. S1 reported to work around the tension by just doing what she does:

> I'm a warrior. I hate those standards... You have, I don't know, 10, 20 kids, 20 groups of the same age level and everyone needs to be seeing the same amount of a cup of coffee with the same amount of sugar—and I like coffee with milk.

S1 describes a one-size-fits-all approach to instruction within a school that emphasizes testing culture, and she sees herself as doing something different, and will continue to do so because she is a warrior, and will fight for what she believes. It is interesting to note that someone who has successfully integrated coding in a physics classroom feels that she is working against the system rather than feeling supported by it.

S1 summary. S1 reported to use coding in order to mediate students' understanding of physics content. She described her pedagogical focus on engaging students in making their processes visible during problem solving, and implemented coding and computational thinking as a means of supporting students in that endeavor. Consequently, S1 identified the negotiation between making student process visible (pedagogical knowledge), understanding density (content), and programming (technology), thereby supporting the literature for PaCK and TPaCK (Koehler, Mishra, & Yahya, 2007; Shulman, 1986, 1987). In addition, S1 defines a tension in her system as she integrates coding in the classroom; while testing culture supports solutions over process, she uses code to make the process more important than the solution. S1 finds a work-around for this tension by ignoring testing culture and pushing her agenda in the classroom.

S3 and instruction. S1 is a math and science special education co-teacher who reported to use Python to encourage students to check their work and to expedite computations. When asked how he integrates coding in the classroom, S3 replied:

> Well you get these a lot of kids get these worksheets they get-- here's 20 examples of different radius of circles. Calculate the area, circumference. So they have to punch through and type in PI R squared or whatever the equation is. So they have to see that kind of thing is something that lends itself to a computer, making it a lot more, I said accurate and a lot faster… having the opportunity to actually create your own tool specific to the task in the classroom is something totally different for them. I mean, I've shown kids how that they can type in equations in the browser window and how that can do something for them… That's like sort of an aha moment. But actually taking a program and actually writing something to do with your need, that was where I see it as being is really kind of a cool adjunct to what you're learning.

Like S1, S3 perceives a direct link between content and coding. While S1 uses computational thinking and algorithms to make students' processing visible, S3 uses the computational power of computers to reinforce mathematics performed in a chemistry or geometry course. Given a worksheet, S3 noted that if a student can create a tool for expediting the calculations, they have internalized the mathematics required. Additionally, S3 mentioned that students create tools to meet their specific needs, situating the coding on a meaningful activity.

S3 and perceived value of coding. S3 reported to contextualize coding in a chemistry or mathematics class in order to engage students in designing their own tools in order to expedite computations and internalize formulas and procedures in mathematics. When asked what value he sees in this endeavor, S3 responded:

> I mean, kids are consumers of this stuff, but they don't look upon themselves as creators. And I want to have them look at the computer, just a tool that allows them to create. It could be something as simple as helping them look at their data in the chemistry class and say, okay, I made my numbers correct. Or if I have a fairly long worksheet in physics or a worksheet in geometry or algebra, a little computer program could help me do the work a lot more accurately and a lot more efficiently. And also, I'm of the opinion that if a student can write a computer program to express the equation they have to get there, it's just another way of looking at the data and having a way of saying, I know how X and Y are related. I know how the data can yield the results that I need. And if they can envision a computer program that can translate that equation, they really have to have ownership of it. So, I think it's a way to reinforce the key concept that we're trying to teach them in that content area.

Not only did S3 refer to coding as a means of creation for students, but also he reported that he wants students to see themselves as creators, not just consumers of technology. Like S2, S3 uses code as a means of identity formation. But unlike S2, who reported to want her students to see themselves as techno-agriculturalists, S3 described his goal as developing content creators. S3 also mentioned that program creation implies ownership of knowledge. Creating a program to calculate the area of circle means that a student truly understands that process, and can generalize their knowledge to any parameter for the radius. Furthermore, unlike S2 who uses code in the classroom to promote technological literacy, S3 focuses on technological competence. There is an intended purpose for these programs, to calculate something.

S3 and perceived impact on students. Despite focusing on identity formation, S3 reported that a good portion of students do not like coding:

> There are students that initially, there's a good percentage to say, God, it's not for me. Programming is not for me… So there's that initial problem with people saying, I don't see myself as being a person who does coding. So I think that's a sort of a low-level kind of thing that people think of when they think about coding. They think about Bill Gates, Mark Zuckerberg, Musk. They think of these famous people who basically look like me, you know, white men… And so I try to dispel that kind of myth that there are people who don't look like me, who are very successful in programming, in engineering… And so that's part of it. Again, giving them baby steps. When I give people programs to look at and when we write programs, they're very short… So I try to give them things that are very simple. That gives them that feeling of accomplishment.

S3 acknowledged that not many students appreciate coding, and noted that identity formation is at fault. Students do not see themselves as programmers since the famous coders tend to be white males. S3 reported to try to dispel these myths, and gives his students short and basic programs in order to give them early and frequent feelings of accomplishment.

S3 summary. S3 reported that special education is his second career; his first career was in IT. Furthermore, S3 initially began using code in his class because of his background and desire to move students toward seeing themselves as creators rather than consumers of technology. Interestingly, S3 did not attend training that initiated his desire to integrate coding in the classroom. Nonetheless, S3 reported to give students sample code from which to build (pedagogy), and distributes his code to students via repl.it (technology) in order for students to internalize the mathematical procedures inherent in a chemistry or math class (content). Again, a subject utilizes TPaCK to mediate the student experience.

S8 and instruction. S8 reported to give middle-school students projects with a list of constraints that determine the rules of their scientific simulations. For example, S8 described the following project:

> We're going to make an energy bar and the energy bar has these constraints. The constraints are it can be only 250 calories, only 70 percent of the calories can be from fat or 40 percent, or whatever it is. Point is there's a bunch of constraints like that. And so spreadsheets are really good way to do that because you can do a lot of what if. Oh, I'm going to put in more sunflower seeds. And that affects this cost. That affects the amount of calories from sugar and fat and whatever. And then at the end, you're doing something which the subject of A) will the bar withstand… a textbook being put on it and not turn into mush and B) Does it taste good or does it taste horrible? So it's a high engagement lab… because now you can do a simulation.

According to S8, students create a spreadsheet with variables and add functions based on the science they are learning. Students can change the values of cells to find values that follow the constraints of the project. Like S7, S8 described a trial and error approach to coding, where students make meaning of the science by exploring how altering one variable changes the other variables in the system. Here, building and modifying a simulation mediates students' understanding of the science standards.

S8 and perceived value of coding. S8 referred to simulations as a tool for scientific exploration. In his words:

> The fact that they build something… I'm explicitly trying to give them a fairly difficult, you know, almost dreary task. And so the spreadsheet as a way out… Intellectually, that's exciting… And they see, oh my God, yes, I can really use this to make my life easier. So again, purely as a tool, not as an end in itself… The engagement there is good again.

Unlike the teachers who use coding as a tool to communicate learning through some constructed artifact, S8 said that he does not view coding and Excel as an end in itself; rather, as a tool for conducting science. Earlier, S8's discussion of building a "scientist identity" in his students was presented. Since simulations are a tool of science, it can be inferred that S8 made a connection between using the tools of science and forming a scientist identity.

S8 and perceived impact on students. Because simulations are a tool for science, S8 perceives his role as forming identities by using the tools of the trade. When asked how coding helps students, S8 replied, "I felt like part of what I'm doing is equipping people with tools. And the direction that you take after you possess this tool is up to you." Similar to others, S8 reported to not want to make future programmers or even future scientists; instead, he perceives his role as giving students a tool that may help shape their identities as scientists.

S8 summary. S8 said, “I don't think it's just me leading the horse to water, but trying to allow them to flourish in a fashion that they, you know, self-directed fashion.” S8 reported to use simulations as a means of building an environment and experience that allow individuals to flourish by engaging them in the tools of science in order to build their identity as scientists. Here, coding is a tool that scientists use for inquiry, and S8 reported to give students access to that tool.

Coding as a Hybrid

One subject reported to use coding as both an encapsulating environment for problem solving as well as a method of mediating course content. Interestingly, analysis leads to the conclusion that the content being taught dictated which pedagogical schema they chose. In general, content in the humanities was encapsulated in an environment designed for communication and self-expression while STEM-based content was directly mediated by the coding. The following subject reported to integrate coding into their classroom using multiple modalities.

S4 and instruction. S4 reported to use coding mainly as a vehicle for students to communicate their learning of reading and writing. When asked how she integrates code in the classroom, S4 responded,

> The number one way I use Scratch is for storytelling, whether they're retelling a story or creating their own story or retelling content—like if they're taking what they've learned and creating a little animation or a video or a game.

Here, S4 reported to use Scratch as an encapsulating vehicle for students to create their own narrative or story in a creative way, situated on communicating their understanding of the content learned in the class. S4’s use of Scratch aligns with the literature regarding the language, as the developers wanted any child, given any background or experiences, to program their own stories, simulations, animations, and interactive stories, and to share them with anyone else in the world (Resnick et al., 2009).

Because S4 is an elementary level teacher, she teaches various subjects. Interestingly, when she described a project in her math and science classes that utilized code, S4 reported that coding directly mediated students' understanding of the math and science concepts:

> This year we actually got a grant for Sphero robots and we made an entire booklet that the kids worked through based on geometry, measuring angles, getting into acute and obtuse angles and stuff. And we had, we just took butcher paper and drew some angles. And so they had to code the Sphero to trace the angles and then they had to create shapes and create paths that the Sphero had to travel on. So then they kind of got to do the coding with a robot and see the actual output while they were actually learning about angles and measurement and, you know, even getting into science with speed and acceleration because they had to figure out, okay, well, if this is 50 centimeters long. When we code Sphero, you don't say 50 centimeters long. You have to do it by speed and the time. And so then they were figuring out speed versus time and equaling the distance. And so we made charts and graphs based on speed and acceleration and all of that.

Unlike S4's treatment of code in her reading and writing projects, the Sphero is not being used as a vehicle for communication, but rather as a tool for inquiry as students engage in hands-on, student-centered learning. S4 mentioned the connections that students make in their inquiry such as translating distance to speed and time due to the input parameters of the Sphero. S4 also noted the connection she drew between the code and traditional graphs and charts. This project is not rooted in just mathematics, but connects physics to mathematics, and does so in an investigative and inductive way as students produce shapes using algorithms rather than traditional formulas.

S4 and perceived value of coding. S4 reported to use coding in her humanities classes as a means of allowing students to communicate their learning in a creative way while problem solving. When asked why she uses coding, S4 replied:

> It's not so much the coding aspect of it… they're going through the process of solving a problem with technology… It's not just the physical coding, there's so much in that realm of just helping them understand creativity and problem solving… They've gotten different tools and they can decide what they want to create.

S4 reported to value personalization, creativity, and student choice as they develop tools and media to communicate their learning. Interestingly, Jenkins et al. (2006) define innovative media creation using technical skills as a measure of youth's digital literacy. Considering that all subjects reported to use coding as a tool, S4 implements coding in her humanities classes as a tool like any other educational tool we use to engage students within meaningful exploration. Within her humanities classes, coding aligns with constructivist pedagogies in order to further engage students in exploring content (Koehler et al., 2007; Shulman, 1986, 1987). Programming, then, can be integrated into any class since it can be used as a technological tool that aligns with constructivist pedagogies in order to engage students with meaningful inquiry and problem solving.

With regards to math and science, S4 reported coding as resolving a tension between testing culture and students' understanding of mathematics:

> So that was fourth graders and we actually did that because we didn't have enough time to get through all our math in the year before state assessments. And so we actually

> integrated it with our science time and made it a computer science, computational thinking mixture of math and science and focused on the distance versus time and all of that.

S4 reported to decompartmentalize mathematical and physics knowledge by integrating a geometry lesson in her science class while also hitting science standards related to speed and distance. She stated that due to the lack of time to get through all of the math standards in preparation for state exams, she had to find more creative methods of giving students extra time with math standards. Interestingly, S4 recounted that computational thinking and coding supported student acquisition of both the physics and math standards.

S4 and perceived impact on students. In her humanities classes, S4 reported to use coding as a tool for students to communicate their learning, thereby transferring the act of coding to a digital or computational literacy that all future citizens should possess. When asked what impact coding has on students, S4 replied:

> It's really a mindset. It's really preparing kids for a whole 21st century mindset. So to me now it's bigger than just coding itself. It's the whole computational thinking and that whole mindset of how can we solve problems using technology. And so I think that's been my biggest shift. And I think a lot of that was just I didn't really know what I was getting myself into when I first started with code. Like, I was just like, oh, this is cool. These cute little robots or, you know, Scratch Junior is fun. But I think seeing the bigger picture is that I think in the beginning I kind of had this idea like, oh, everybody is going to have to be able to code when they're older. And I don't really do that anymore. I see more the bigger picture that if you have a mindset, computational thinking mindset or you have some basic knowledge of code, you're gonna be more marketable in the health

field, in the movie field, you know, the film industry. I mean, it opens your doors in every industry, not just the coding IT industry.

S4 supported the earlier claim of valuing computational literacy by describing a 21st century mindset of solving problems with technology. S4 described shifting from using code to engage students because it is a fun thing to do to imagining code and computational thinking as important skills that are required for future activity. S4 also differentiated learning code to go into the IT field with learning code to better students' participation within any field.

Regarding the math and science content, S4 reported that students enjoyed the lesson and hypothesized that they understood the content working with Spheros better than if they had been taught the content using the traditional books and activities: "They loved that. And I think they understood angles and measurement better than if we had done it the traditional way with our math curriculum that we have." Interestingly, S4 describes the impact on students within STEM fields differently than she does for humanities fields. While coding in humanities increases students' computational literacy, coding in math and science directly mediates students' understanding of the concepts and decompartmentalizes knowledge.

S4 summary. S4 is interesting because she integrates coding differently within different subject areas, and her implementations align with other subjects' reported experiences and perceptions. In her humanities classes, S4 reported to integrate coding as a means of giving students a new vehicle for communicating their understanding of the content, thereby elevating code to a new digital literacy. In her math and science classes, S4 reported to integrate code as a means of directly mediating students' understanding of the concepts and decompartmentalizing knowledge between math and science through problem solving and inquiry. In other words, depending on the content, coding aligns with different pedagogical schema and methodologies.

This result further amplifies the importance of considering TPaCK when trying to understand how coding mediates students' experiences.

Summary of Pedagogy and Use of Code

Subjects either described using code as an environment for students to communicate their learning of some content or as a means mediating students' understanding of the content standards. Interestingly, all of the all subjects who teach the humanities described integrating code as a medium for communicating ideas and self-expression while four of the five STEM teachers reported to use code as a means of mediating students' understanding of the content standards. Table 4 shows each subject, content taught, described pedagogy, and how that teacher reported to use code in the classroom. S4 is present twice because she reported to use code via both modalities depending on the content.

Table 4

Summary of Instructional Models Based on Content

Informant	Content	Pedagogy	Use of Code
S1	Science - Physics	Content mediation	Computational thinking
S2	Ag. Science	Communication	Narrative or Game
S3	Math, Science	Content mediation	Function for computing
S4	ELA	Communication	Narrative or Game
S4	Math - Geometry	Content mediation	Robotic motion
S5	ELA - Literature	Communication	Robotic art
S6	ELA - Poetry	Communication	Robotic art
S7	History - First peoples	Communication	Narrative or Game
S8	Science - Physics	Content mediation	Simulation

The data present an interesting pattern of teachers negotiating pedagogical decisions based on content.

Teachers who integrate code as a means of content mediation described variety in their use of code; no two informants reported to use code the same way. Teachers who use code as a form of communication and self-expression did not report variety in their use of code; either code was used to develop a narrative or game or code was used to create some sort of museum exhibition or diorama that symbolized student analysis. TPaCK describes the mediation of technological knowledge, pedagogical knowledge, and content knowledge on each other and the decisions teachers make in the classroom (Koehler & Mishra, 2005; Mishra & Koehler, 2006).

Assessment

Based on the reporting of the informants, the way in which teachers integrate coding also determines how student code is assessed. Informants who described using code as a form of communication and self-expression use a binary rubric with tasks to complete in the project, and coding is embedded within those tasks. Subjects who reported to used coding as a mediator for content assessed code in a variety of ways, situated on how the content is being assessed.

S2, S4, S5, S6, and S7 reported to integrate coding into their classes as an overarching project in which students communicate their learning of some material. Interestingly, they all described their grading scheme as a rubric that students receive before starting the project, with specific functions, tasks, and features that must be present on their projects, and each subtask is graded on a binary scale. For example, S2 reported:

> I put up like a rubric that this is what I want, like you need to have the example on the other one, four sliding screens, two working buttons per page, like I put it that way. And that's how I assess, because if it all works, then they're good.

S2 described grading a project in which students create an app in Scratch that displays information on different screens, making the presentation interactive and engaging. She noted that there must be four screens that slide and two buttons so that the user can go back and forwards in the presentation. S2 sent the rubric for this project after the interview. S2 wrote:

> Thunakable app building. Topic of your choice. Must include: 3 changing screens (slideshow), 2 pictures per screen, buttons that when clicked they tell about the images, final screen has a 3 question quiz. (4 screens total). Text needs to be large enough to read. Background color should look nice with the pics you choose. If animals, add a sound file. This needs to be 8th grade quality. Your user needs to learn something new from your app. At least one button must speak when clicked.

S2 emphasized that if the app works correctly, then the code is good; that is, she did not report to grade the style, complexity, or accuracy of the code itself. Rather, the code is graded based on its functionality in the larger project.

S4 made a more explicit connection between a project-based learning classroom, rubrics for projects, and the grading of code. She said, “When I look at my PBL rubric of, you know, did it meet the needs of. Did your museum exhibit engage your audience? So in a sense, but not actually assessing the code itself.” Again, S4 described a rubric with features that need to present in the project, but code is not assessed; rather, the functionality of code is assessed.

Not all subjects graded rubrics on a binary scale. For example, S5 described various uses of a servo motor. She said, “There’s a section of the rubric you know related to mechanisms. Did you just like have a servo spinning and have something glued on top of it or did you use the servo in some clever way?” S5 described a portion of a rubric related to the mechanics of a robotic art project, and students needed to include a servo motor in order to create movement in the installation. All teachers who reported to use coding as a means of communication situated learning within a larger project. Interestingly, all of these teachers (except S7 who said she does not assess projects) reported to give students a rubric, and to assess code in terms of its functionality rather than its accuracy or style.

Informants who reported to use code as a tool for content mediation did not necessarily embed coding within larger projects; rather, coding was an integral part of students' inquiry and process. As such, these teachers described assessment in terms of the assessments they normally give in their classes. For example, S1 reported to focus on process over product, and grades students based on their steps. When asked if she assesses code, S1 replied, "If they have a word problem like density or speed or acceleration, they need to show me this. They need to show the steps so at the end a word problem is not two points." S1 described her use of code as engaging students in computational thinking to decompose problems in science. To her, problem solving is not just two points, a beginning and an answer; rather, it's a process, and computational thinking is embedded within that process. S1 did not share exactly how she grades; just that she grades their steps.

Similarly, S3 reported to assign short Python programs to support students' calculations on math and science homework. When asked if he assesses student code, he replied:

> When you're a co teacher, you don't want to make your one day there very different from the rest of your days there. There are times in chemistry where I've taught with somebody for two or three times where I have an assignment and she'll say, just look, you go grade this, I'll enter it for them. So it just becomes a kind of a lab assignment or another homework assignment.

S3 described coding assignments like any other homework assignment they may have in the class. As such, he noted that it goes in the gradebook like a lab assignment or any other homework assignment. Again, the grading of code is contextualized within the parameters of how content is already graded in that environment.

Not only does there exist a relationship between how informants reported their use of code and the content that they teach, but also, there is a third factor that mediates teacher activity: assessment. Table 5 summarizes the reporting of informants, displaying their content area, the activity that was graded in their reporting, and the assessment given.

Table 5

Summary of Reported Projects with Their Assessment Type

	Content	Activity	Assessment
S1	Science	Computational thinking embedded in inquiry	Process and steps are assessed
S2	Ag. Science	Scratch project	Binary rubric, where code is embedded in tasks
S3	Math, Science	Python program to help calculate	Counts like a homework assignment
S4	ELA	Museum exhibit	Binary rubric, where code is embedded in tasks
S5	ELA - Literature	Robotic art	Binary rubric, where code is embedded in tasks
S6	ELA - Poetry	Robotic art	Binary rubric, where code is embedded in tasks
S7	History - First peoples	Scratch project	No assessment
S8	Science	Simulation	N/A

S1 and S3 both teach within a STEM field, and reported to use coding as a mediating tool for content acquisition. Interestingly, they both embed code assessment within the norms of their classroom; that is, code was reported to be assessed in accordance with their previous assessment schema. Most of the informants who reported to use coding within larger projects for communicating learning such as S2, S4, S5, and S6 described grading on a rubric, where code is not explicitly addressed, but is addressed in terms of larger functionality. It appears that generally, content dictates how code is used in the classroom, which then determines the assessment style for code.

Teacher Activity: Perceived Identity of Self

When asked what advice they would give to others who wanted to integrate code in their class, informants provided advice based on their experiences and perceptions. Interestingly, informants gave very similar advice related to the teachers' sense of identity and role in the classroom. For example, while describing how other teachers deal with coding, S1 said,

> Sometimes they take workshops and they didn't use it at all in classroom and then that is very sad because they don't want to change their comfort zone so they don't want to explore, they don't want to get involved because that will be more responsibility.

S1 noted that integrating code in the class removes teachers from their comfort zones, but did not specify reasons for the discomfort other than the fear of something new. Other informants continued to discuss comfort, and added further reasons for why a teacher might feel uncomfortable integrating code into their class.

S2 insinuated that the lack of comfort could stem from lack of knowledge of coding. She said, "Even if teachers aren't that comfortable. Just try it out. And if they're really comfortable with just teaching ELA, well integrate Scratch. You know, let the kids recreate a scene, something just to get them comfortable." Not understanding how to program or programming environments means that teachers would not know how to best integrate coding with their content knowledge and pedagogical knowledge. S2 recommended that teachers start small, such as a project in Scratch, and work their way from there to ease into coding. S3 provided additional advice about becoming more comfortable: "Find the one place that you know this could be successful… and keep it simple." S2 and S3 both refer to coding knowledge as a barrier for teachers to begin coding in their classrooms, and advise teachers to start small, start simply, and keep adding pieces over time. Additionally, S8 acknowledged the anxiety that teachers may feel, and provided advice for working with teachers. He said that when working with teachers, it is best "not to convey information, but give them an experience which overcomes the activation energy, the emotional hump, preventing them from getting involved." S8 noted that experiential learning can move teachers beyond their initial trepidation.

Other informants reported that teachers do not need to know all of coding, and stated that there are ways to move teachers beyond the initial hump. For example, S4 reported:

> I would also say don't be afraid to give it to your students without knowing everything. Most kids are very tech savvy-- you just give them a drop and give them time and they will wow you. Like they will just take off. And I think a lot of teachers are afraid to kind of unleash that. They're afraid that they won't be able to control it. They're afraid that the kids will get into something that they don't understand or they don't know. And I think that's the beauty of it is just then then the kids can't depend on you… I literally cannot answer their questions. Sometimes I'm like, I don't know. And it forces them to have to solve that problem on their own.

S4 stated that not only should teachers not fear not knowing how to code, but they should embrace it because students will no longer depend on the teacher and will be forced to seek their own answers. S4 did admit that there is some chaos that many will want to control, but her advice was to allow the chaos to unleash and have faith that students will eventually figure out their own problems. S5's reporting agreed with S4 in that teachers need to navigate some sort of chaos in the classroom, whether it's the chaos of not knowing how to help a student or the chaos of classroom management since students are working independently and at different rates.

S6 provided some context to the chaos in which S5 alluded. When asked what instructional modifications she has made in her class since she began coding, S6 said:

> When they were doing this project, my room is—it doesn't look like a traditional classroom, that is for sure, because they are just everywhere. They have materials everywhere. You know, they're not just sitting at their desk and working on something. They're in groups. They're on the floor. They're painting. And, you know, you have to kind of let go… They have to be able to move around. This isn't just sit and listen to a teacher lecture. Basically, I explain the project to them at the beginning. I give them a rubric. I give them the materials that they need if they need help with coding and the Snap programming. Other than that, they're kind of on their own and I'm just kind of monitoring and helping out when I'm needed. So it's my classroom definitely looks different during that project than it does. Let's say if I'm just, you know, teaching a novel or teaching grammar.

S6 described a space in which students are working in all parts of the room, engaged in different activities, and in a state of making. In fact, S6 perfectly described a makerspace in that students are engaged in a hands-on, manufacturing approach to learning (Blikstein, 2013). S6 then differentiated between an instructionist vision of teaching, where the teacher lectures to the class and a constructionist mode of learning, where the students make artifacts. It is interesting to note that S6 describes her shift to a constructionist classroom as a natural consequence of the types of projects that coding affords her and her students.

Not only do teachers not need to know everything about coding, but some informants stated that existing in a state of not knowing can be useful as a new form of teacher identity: an identity of co-learner. S7 said:

> One thing that I suggest is to try to be okay not always necessarily being an expert, but being able to introduce this to students, and I think an offering to help is… I'll do it with you, playing with you.

S7 stated that not only does she consider herself to not be an expert in coding, but also she reported to assume the role of a learner with the student, playing with the coding along with the student as a form of modeling and mutual inquiry. This identity of co-learner for the teacher aligns with previous statements from informants regarding the trial and error nature as well as the creative aspect of coding.

Informants reported a shift in perceived identity by integrating code in the classroom. Some informants focused on managing the chaos of a space in which students are constructing artifacts and working at individual rates and on their own tasks. Other informants described the fear of being asked a question outside of their knowledge base, and transitioning toward an identity of co-learner.

Teacher Activity: Training and Support

It is important to note that all of the informants have been successful in their endeavor to integrate coding into their classes; they reported to do it, and they reported to continue doing it in their class. Whatever barriers exist for other teachers that may prohibit them from integrating code have been overcome by the informants. It is therefore worthwhile to analyze the beginning stages of their activity in order to inform teacher preparation programs and computer science education programs of the qualities, methods, and practices required to move teachers beyond their initial fears.

Teachers as Self-Advocates

One common theme in the reporting of informants is taking the initiative to try new things. For example, S1 reported to apply to a professional development for a new computer science course, and after being introduced to programming, taught herself more. She said:

> I entered into our computer science principle pilot teachers so I learned how to use Snap. That is like a version of Scratch and then I self-taught in Python. So I start playing with Python and I showed them how to use Python.

S1 reported to take the initiative to apply to a specialized program, and then taught herself a syntactical language after being exposed to block-based programming. S2 reported a similar a similar experience. When asked how she started using code in her class, she said, "I had no training on any computer science stuff, so I learned everything, just figuring it out." S2 described an inquisitiveness and desire to continue trying new things. She also noted that the Twitter community was instrumental in providing her with resources and places to go. S2, then, reported to self-advocate, and was supported by a community within social media. S8 also cited self-advocacy situated within a community of practitioners. S8 said, "Took it upon myself. I'm pretty active in the science teacher convention world, if you will. Sometimes I present, but I tend to go to the science teacher conventions." Rather than being supported by Twitter, S8 mentioned that he was supported by science teacher conventions, and took it upon himself to start utilizing code in the classroom.

Although informants described themselves as self-advocates, many were supported by a community of like-minded individuals. For example, S4 explained that she did not have a community of support at her work site, and like S2, used Twitter:

> And just having that network of people that I can ask questions because I don't have anybody in my building to ask questions… And I don't feel like I even have a single teacher that I can reach up to for questions… And then even on Twitter, I've gotten involved and kind of followed some certain people that are posting things and doing things in their classroom.

S4 described a combination of initiative on her part and curating ideas from others on Twitter as her support system for coding in the classroom. S4 was not the only informant who felt unsupported at their local site; however, social media tended to be a venue for curating and forming ideas for the classroom.

Systemic Support

Not all informants explicitly described themselves as self-advocates; however, they did describe their environment as being highly supportive of implementing code in the classroom. Three informants described Pennsylvania as an area with rich professional development focused on STEM and computer science in particular. Interestingly, all three use the BirdBrain Technology robot kits from Carnegie Mellon, a University in Pittsburgh. S5 noted:

> Pittsburgh for teachers has tons of training. You don't have to pay for things like that. And there is a technology bent to them. I take a ton of training all over the city. And programming things are often popping up.

S5 explained that she began using code after several trainings, when she felt more comfortable with the technology. S5's description of Pittsburgh as a haven for programming training emphasizes the role of her community on her practice. S7 also noted the richness of professional development in Pittsburgh:

> Pennsylvania is really pushing for computer science for all. And so now there's training for teachers. We went to one just a few people from each district to kind of get us going... And the expectation is that we're bringing back some of these skills.

S7 explained that although there are many trainings, teachers are expected to bring the skills they learn back into their local sites. Furthermore, S5 mentioned that administrators are supporting the integration of technology and computer science in schools. S7 portrayed an area where educational technologists provide professional development for teachers, administrators at schools are supportive of technological integration, and teachers are embedded within a culture of progressive activity.

S6 made explicit the members of the community who supportive her activity as well as the division of labor within the system. When asked how she started using code in the class, she explained:

> Our school started to have a real push with the coding and using the hummingbird kits that are made by BirdBrain Technologies. We did a lot of trainings at our school and how to do that… So that's how I got into it. We also have a teacher at our school who's very much involved and tied in with the BirdBrain Technologies… And she actually kind of asked me if I would be interested in doing something in my class with the Hummingbird kits. And at first I was a little like, don't know what I would do in my class with it. And then I started to do some research online. And I came across other people who were doing poetry projects using the Hummingbird kits.

S6 described the actors within her community: the technologists at BirdBrain Technologies who designed educational robots; the professional development facilitators who brought trained teachers at her school site to use the robot kits; the administrators who supported the professional development; and the expert teacher who worked with her one-on-one. Each member of the community divided labor to embolden S6 to move beyond her initial fears of coding. S6 did not describe herself as a techie, and expressed her initial fears at the prospect of integrating code into her classroom. For S5, S6, and S7, who did not describe themselves as self-advocates, their community and the educational tools designed for learning supported their practice.

Research Question: Experiences of K-12 Teachers Who Recontextualize Code

The purpose of this exploratory qualitative study was to understand the perspectives and lived experiences of K-12 educators who have intentionally integrated programming into another K-12 field. Specifically, this study sought to answer the following question: How do K-12 teachers describe their experiences and perceive their efforts to redefine programming in a curricular landscape other than a computer science class?

Common Experiences

Informants shared common experiences in their endeavor to recontextualize code, which include: treating code as a digital literacy; integrating coding into their mediation of technological, pedagogical, and content knowledge; successfully moving beyond tensions within their system of practice; shifting classroom activities toward a constructionist approach; playing with code to develop engaging activities rather than using prescribed curricula; and engaging students in interdisciplinary problem solving.

Code literacy. Each of the humanities teachers described coding as an encapsulating environment for problem solving, creativity, and communication. Coding as a method of communication aligns with research that places coding as a digital literacy, where creative construction is part of everyday endeavors (Jenkins et al., 2006). Furthermore, informants who use coding as a form of communication tend to assess code as an embedded entity within rubric-based grading schemes, which further aligns with the digital literacy literature. Code is interpreted as a tool that allows for the creation of media, which is a form of human expression and communication.

TPaCK. All eight informants described computer programming as an instructional tool; however the four STEM teachers (S1, S3, S4, and S8) shared the experience of using coding as a mediating tool for content acquisition. S1 and S8 reported to use Excel spreadsheets and Python to create simulations for in-class inquiries. S3 reported to use Python as a means of developing basic programs for calculating and checking errors on worksheets. S4 reported to use the Sphero to bridge Geometry and Physics. This aligns with research that describes the mediating role of technological, pedagogical, and content knowledge on teachers' practice (Koehler et al., 2007; Shulman, 1986, 1987). Furthermore, these informants reported to assess code according to the previously defined schema they use in their classes; that is, coding is a tool that engages students in knowledge creation, and was assessed as a process of knowledge creation.

Cultural-historical activity theory. All of the informants were successful in their recontextualization of coding. They chose programming languages and environments designed for students. These tools mediated their experience as teachers, often forming and reforming their identities from lecturer to co-learner. Although many informants described testing culture as a challenge, all informants were supported by their administrators and coworkers, or at least their online communities. Informants described rich opportunities for professional development in their local communities. The main source of tension reported by teachers was technology not working; some used those days as opportunities for students to personalize their projects while others reported to use peer pedagogies that required students to share devices. A few of the informants described students as perceiving coding to be exterior to their notion of self, and did not care to participate in coding activities, and provided alternative activities within the projects for those students. In all, there were not many tensions reported by informants.

Constructionism. All informants, except S3, described how their perceived identities as teachers formed and transformed during the process of integrating code into their classrooms. Informants described a certain chaos in their classes, where students are making and constructing in various areas of the room, and groups are focused on different tasks at the same time, which is divergent from how they describe their classes when they do not code. Some informants reported that not knowing everything is beneficial to creating a culture of inquiry, and perceive themselves as co-learners rather than knowledge-givers. Most informants described their classroom as constructivist, even constructionist, while coding, whereas they reported to instruct when they do not code.

Building teachers. The literature suggests that computer science education historically builds curriculum rather than empowering teachers to create their own experiences and activities (see Chapter Two). Interestingly, each of the eight informants described projects and activities that they developed over time after playing with the programming environment and learning from others in their communities. Each informant described a level of play and *messing around* with technology, followed by trial and error experiences implementing code with students. Over time, teachers built enough confidence to design and implement interesting and engaging projects.

Interdisciplinary problem solving. Due to the nature of recontextualizing code, the activities and projects that informants described were interdisciplinary, decompartmentalizing knowledge. Reported activities required students to perform computational thinking, mathematics, spatial reasoning, logic, and systems thinking while engaging in course content.

Teachers described a certain amount of chaos that resulted from the interdisciplinary nature of recontextualizing code; students approach problems from a variety of disciplines, requiring the teacher to adapt and improvise when supporting struggling students. In part, this uncertainty of student challenges required teachers to modify their role in the classroom to a co-learner identity.

Unique Experiences

In addition to shared experiences, some informants described unique experiences worth noting. S2 experienced the masculine occupation culture of computing when she asked to partner with the on-site computer science teacher, S4 utilized coding as both a mechanism for communication as well as mediating tool, and S1 left her previous school because admin did not support her use of computers in the classroom.

Masculine occupational culture of computing. S2 described trying to build a professional relationship with the on-site computer science teacher, who in turn rejected the notion of Scratch as a real programming language and laughed at student work, mocking their projects as not real computer science. The need to distinguish real computer science from other forms of computing is an instance of feelings of superiority over users, which is embedded within the masculine occupational culture of computing (see Chapter Two). Furthermore, the scenario illustrates hostility toward S2, and aggression toward women is another example of the occupational culture of computing. The scenario presented by S2 demonstrates an example of how the occupational culture of computing manifests within the K-12 space and negatively impacts those who use the tools of computing for learning.

TPaCK. S4 described herself as utilizing code as an encompassing environment for communication and self-expression; however, she did describe one project in which she used the Sphero in a Physics class as a means of teaching Geometry. Interestingly, S4 modified her use of coding depending on the content and how coding related to her content standards. When it was not obvious how code could mediate students' understanding of content, S4 used coding as a mechanism for communicating ideas, much like presentation software, and yet, S4 described an instance in which she used coding as a direct mediation of Geometry and Physics content. This implies that S4 consciously determined when to use code as a mediating tool and when to use it as a mechanism for communication.

Cultural-historical activity theory. S1 described a richly developed science program where she recontextualized programming via Python and Excel to run simulations in Puerto Rico; however, she was told by her administrator in Texas that computers do not increase test scores, and she was not allowed to use programming. The high stakes state testing focused administrators on a single task of increasing scores, and innovation such as recontextualizing code decreased for more traditional methods of teaching. Because autonomy and engaging activities were important to S1, she left that school in favor of another that allows her to recontextualize code. In this case, the rules governing the system dictated S1's administrator's focus, which in turn impeded her objective of recontextualizing code.

The Lived Experience of Teachers Recontextualized Coding Outside CS

Ultimately, the informants described ingenious methods of using code as a tool in their classrooms, and through their activity, have expanded what it means to do code. Computer science education has tended to be exclusionary and reserved for few (Fisher & Margolis, 2002; Margolis et al., 2008), and the informants described rich environments where all students participate in coding, and find meaning situated on their inquiry into content. Furthermore, most informants described a makerspace quality to their classroom, where students use a variety of tools and technologies to construct artifacts. Although informants described challenges that hindered their progress, they were all successful in implementing coding in non-computational classes, and their experiences and perceptions of coding can provide constructive feedback for teacher preparation programs as well as the computer science education field as a whole. There were two models of recontextualizing code: teachers who use programming and programming environments as encapsulating vehicles for student communication and self-expression; and teachers who weave programming into content standards in order to mediate students' understanding of content. The former were called *Communicators*, as they used coding as a mechanism for student to communicate their understanding of learned content, while the latter were called *Weavers*, describing the manner in which they integrate programming into the content of the course.

Communicators were not previously trained in computer science; however, they all attended trainings where they learned a specific technology (like Scratch or BirdBrain robots) or they were self-advocates and sought information on integrating code into their class via social media platforms such as Twitter. Communicators had significant support at their school sites; administration supported recontextualization, and sometimes even initiated teachers' practice. If a teacher was geographically isolated, he/she used social media as a means of supporting his/her craft and trying new activities. Communicators described their classroom as chaotic, as groups of students are engaged in different tasks across the room. Interestingly, each Communicator noted that it is impossible to foresee all challenges that students will face, and often did not know the answers to students' inquiries, since students use diverse codes depending on their projects and interests. Consequently, teachers shifted from an instructionist identity to that of a co-learner, figuring out challenges with students. Furthermore, each of the recontextualization activities described by Communicators were constructionist. Communicators use project-based learning activities for their recontextualization, and provided students with binary rubrics that contain the features that must be present in their projects. Code is not explicitly assessed; rather, code is embedded within the features of the rubric, ensuring that students use specific algorithms and data structures in their projects. Communicators perceived the value of recontextualizing code to be increased student engagement, access to a skill that will be useful in the future, and deeper engagement with course content.

Alternatively, Weavers did not report to learn about coding from professional development opportunities; rather, they described themselves as highly engaged in local conferences for math and science, where they shared knowledge with other practitioners and learned about methods of recontextualizing code. Two of the Weavers gave their students projects that required building and playing with a simulation, one Weaver had his students build simple algorithms in Python to check their work, and one Weaver used Spheros to bridge physics and geometry. Each of the Weavers used the fact that data manipulation and analysis is a cross-cutting concept between math, science, and computer science as a means of integrating coding into content standards. Data manipulation and analysis is a basic skill required for mathematical and scientific modeling, and coding became a tool for student inquiry into modeling. The Weavers who used Microsoft Excel and Python for their modeling were challenged by introducing sufficient basics of code to support future student exploration and inquiry without turning students off coding. Additionally, one Weaver in particular was told by her administration that she could not recontextualize due to the school’s focus on state testing and she changed schools in order to continue her practice. Since coding was built into projects that already existed in their classes, students were assessed on their process of discovery, and although coding was not explicitly assessed, students could use their coding in their discussion of their process. Weavers perceived student identity to be a barrier for their engagement in math and science and used recontextualization as a vehicle for identity formation and transformation as students engage in hands-on learning. All Weavers described themselves as instructionist; however, all of their recontextualization activities were constructivist in nature.

Chapter Five: Discussion

Computer science education in the United States is in crisis. Credentials for teaching computer science are rare, computer science is not a graduation requirement in most schools, and the K-12 system suffers from a lack of computer science teachers. The few computer science teachers that currently exist do not come from industry; rather, they generally start in math, science, English, or history, and transition to computer science due to state mandates or program changes in schools. Because teachers are often not trained in computer science content, computer science education has relied on tools and curricula developed by those in IT industry to mediate teachers' practice as well as students' experiences. The masculine occupational culture of computing transferred from industry to the K-12 educational space via the tools and community members within practitioners' systems of activity. The activity of programming, as it occurs in school settings, is embedded within a particular and traditional cultural milieu that research suggests has an exclusionary impact on students, which explains the low numbers of women and students of color who take AP Computer Science A.

One solution to the computer science education crisis is to circumvent stand-alone computer science classes and to integrate computer science into traditional, general education classes. Increasingly, teachers in subject matters other than computer science or programming courses are including coding activities in their coursework. This sort of pedagogical context for coding creates opportunities for students to view and experience programming as a meaningful and valuable skill, embedded within learning contexts that are not exclusionary. While still relatively uncommon, the efforts of teachers in subject matters other than computer science offer insight into the ways in which programming can be integrated into meaningful curricular contexts. It is then worthwhile to look at the activity of computer programming within non-computational spaces in order to understand how teachers integrate coding into their classrooms and their experiences in doing so.

This study focused on formal K-12 spaces where computer programming is intentionally integrated into non-computational fields in order to give new meaning to those engaged in the practice of programming. More specifically, the purpose of this exploratory qualitative study was to understand the perspectives and lived experiences of K-12 educators who have intentionally integrated programming into another K-12 field. Eight teachers across the United States were interviewed to understand their experiences and perceptions in recontextualizing code. The results of this study have considerable implications for schools of education as per training and support, practitioners in terms of curriculum and instruction, and students in terms of opportunity and access to a skill historically open to few.

Eight informants were interviewed in this study; five teach humanities, and four teach math and science classes (S4 teaches both). In terms of how teachers used code in their classrooms, the eight informants can be considered in two groups: five teachers used programming as an encapsulating environment for communicating learned materials; and four teachers used programming as a direct mediation of content standards (S4 uses coding in both modalities). For the purposes of this study, the group that used coding within larger projects designed to engage students in developing narrative for communication was labelled *Communicators*. An example of a Communicator activity is students creating an interactive museum display in order to communicate their learning of some material. The four teachers who reported to integrate code into their math and science classes as a means of mediating content were labelled *Weavers*. An example of a Weaver activity is asking students to design a new nutritional bar, and use a simulation in Microsoft Excel to change different amounts of product and determine how those changes affect the nutritional values, cost, and specs of the food product. Because S4 reported to integrate coding in both ways, she will be considered a Communicator when she designed projects where students build a narrative of their learning in Scratch, and will be considered a Weaver when she used the Sphero to engage students in Physics and Geometry content.

Implications for Communicators

All teachers can implement Communicator recontextualization. As reported by the informants, recontextualizing code as a Communicator compels: a constructionist, project-based learning environment (even temporary); access to technology; use of a programming language and environment with a low floor and high ceiling such as Scratch; a school culture that supports technology and innovation in the classroom; identifying as a co-learner engaged in inquiry with the students; binary, rubric-based assessments; and involving all members of the community.

Learning environment. All Communicators described structuring a classroom space and learning environment that supports creative, group-based inquiry. Informants portrayed an *energetic chaos* in the learning space as groups work on different tasks in various locations of the room. Informants described initial fear of the *makerspace chaos*, as their preferred classroom management involves more control and less noise; however, they all perceived the chaos positively during the time of the interview.

Technology. Since the informants were successful in their recontextualization of code, they had access to classroom sets of computers and robots with which students could use. Scratch requires a device with Adobe Flash enabled as well as internet access. The Hummingbird Kits require a computer for writing the code as well as a USB cord to download the code onto the individual robot. All technology must be powered, which requires some organizational model for storing and powering the technologies. Communicators reported to have support from the school technology staff to support them in managing, maintaining, and troubleshooting technology; however, they did note that technology issues often occurred, and were frustrating. Despite the challenges, each teacher found some system that worked for them given the constraints of their classroom space and the technology afforded to them by the school.

Programming language. Interestingly, all of the Communicators chose Scratch as the programming language and environment. *Scratch* was designed to alleviate much of the frustrations associated with learning a programming language by using blocks to provide a low floor and high ceiling (Resnick, Silverman et al., 2009). Informants reported to choose Scratch due to its ease of use and reduced introductory instructional time. Furthermore, Scratch's focus on media creation, specifically as narratives (see Chapter Two), aligns the language to the overall student activity implemented by informants; that is, building a narrative of learned content is embedded within the design philosophy of Scratch. Although not necessarily inferred from the data, it is interesting to note that advances in computer science education such as Scratch may have enabled Communicators due to its ease of use and focus on building narratives. Similarly, the introduction of robot kits from BirdBrain Technologies have enabled more schools to engage in robotics due to their low cost and low floor.

School culture. Each of the Communicator informants reported to work in a school environment that supported and even initiated their practice of recontextualizing code. Community members both at the school site as well as in the Twitterverse were instrumental in informants' development as recontextualizers, and provided opportunities for growth and reflection. It should be stated that not all of the informants reported themselves as self-advocated; in fact, S6 initially did not want to recontextualize code, and was instructed to do so by her school administration. A veteran teacher who worked closely with BirdBrain Technologies worked one-on-one with S6 throughout the year, alleviating her initial concerns and trepidation. Overall, the degree of support required by teachers seemed to be directly proportional to their perceived level of *techieness* as well as their perceived level of self-advocacy.

Community. Communicators reported to be significantly supported by their communities. These are teachers who have no background in coding, and yet managed to run coding activities in their classes. Many of the informants were initially trained by private organizations that developed a particular technology (like BirdBrain) or a specific curriculum (like Code.org). Although not explicitly part of the results of this study, it was interesting that the Communicators who discussed the need for equity and access in computer science education were trained by organizations with a grant from the National Science Foundation. Beyond initial training, Communicators were significantly supported either at their school site or by their online community.

Teacher identity. All Communicators reported a distinct change in their instruction as well as their identity as teachers. Informants noted that their usual teaching style is more instructionist, where students listen and the teacher provides information and tests students on that information; however, during Communicator activities, informants reported to perceive themselves as co-learners, investigating challenges alongside students. This was partly due to the open-ended nature of coding, and the inability to know all answers before students begin their inquiries. Informants described a certain control being lifted in order to be able to tell students that they do not necessarily know the answer to a question. Interestingly, informants reported that the co-learner identity helped them communicate better with students.

Assessment. Communicators develop rubrics for their projects, thereby communicating the scope of the project to students. All of the Communicator informants described their rubric as binary, where they look for the presence of certain features and content. Coding is not explicitly assessed; rather, coding is a required process in the development of necessary features. For example, S2 requires at least four screens with back and forward buttons on one of her projects, thereby ensuring that students have inserted event handlers for each of their buttons. Students then are motivated to figure out specifics of coding in order to develop meaningful features in their projects. Interestingly, recontextualization for Communicators situates code within the features required in a rubric, which is a significant divergence from traditional computer science education, where coding is learned without context. Furthermore, informants described projects as open-ended, where students could personalize and insert content meaningful to their sociocultural backgrounds.

The Cultural-Historical Activity of Communicators Recontextualizing Code

Each of the Communicators in this study was successful in recontextualizing code. In fact, the criteria for subjects required success in recontextualizing code. For this reason, there are limited tensions within the system in which these teachers practiced. Let the object of the system for Communicators be recontextualizing code. Then Communicators' practice was mediated by: the tools and symbols of the system; the rules of the system; the community; and the division of labor. Figure 11 shows the diagram of a Communicator's system as they engage in recontextualizing code. Interestingly, the diagram is incredibly similar to the diagram in Chapter Two (see Figure 7); however, the diagram below does not have as many tensions due to fact that each of these teachers were successful, a specific criterion for subjects.

Tools and symbols. In addition to Communicator's beliefs about coding and education, they all chose Scratch as the programming language and environment. Scratch reduced the amount of time and effort teachers needed to introduce and teach specific of coding while facilitating student inquiry. Each of the informants described the space of the classroom as largely affecting their practice as well as how students engaged in the projects. A few of the Communicators relied on online forums and resources such as Twitter, especially if they live in a rural area with limited access to local professional development opportunities.

Rules. Aside from the traditional rules that govern teacher practice such as educational code and the state standards, there were two surprising systemic rules that significantly mediated Communicators: state testing and district policies. The Communicators worked at school sites that allowed innovation and recontextualization despite also focusing on state testing. Unlike S1, who was forced to move schools due to a school's narrow focus on state testing, none of the Communicators were challenged by administrators with a singular focus on increasing scores. Additionally, informants from Pennsylvania initiated their recontextualization due to a state-wide focus on increasing STEM education, which increased the number of opportunities to learn about coding and education.

Community. Communicators benefited from their community—as a resource for training and continued support as well as a means of increasing the meaning of a project. It has already been stated that informants were supported by local professional development facilitators as well as colleagues on their campuses and community members in online forums. Interestingly, teachers also leveraged community members such as parents and other students during projects in order to increase the visibility and meaning of the projects.

Division of labor. In order for a teacher to reach their potential as a recontextualizer, professional development opportunities had to be developed, planned, and executed. Colleagues and online practitioners had to take time to answer individual questions, provide insights, and support in general ways. Teachers had to plan, develop materials for students, setup classrooms and technology, and develop rubrics for assessments. In addition, teachers had to troubleshoot technology and learn with students.

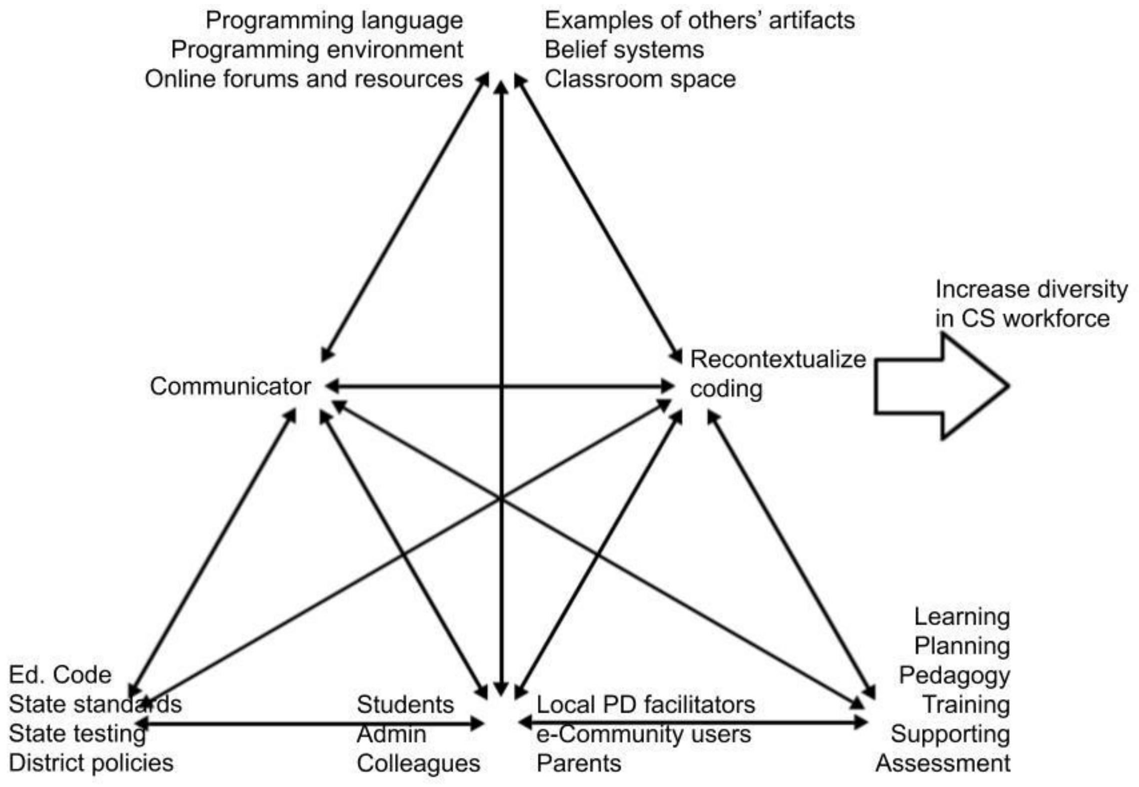


Figure 11. Cultural-historical activity of communicators.

Implications for Weavers

Weavers are more specialized than Communicators; Communicators taught all subjects, but Weavers were specifically math and science teachers. Weavers integrate coding activities into their content standards via the overlap of data storage, data manipulation, and data analysis in math, science, and computer science. For example, two Weavers had students build simulations, where they define variables and functions that affect the larger system, and then manipulate the variables to determine how the system is in turn changed. Another Weaver uses a Sphero in a Physics class to create geometric shapes, and students must translate speed and time into distance in order to create the shapes.

Computational thinking is specifically addressed in both the Common Core Standards for mathematics as well as the Next Generation Science Standards, and engineering and computer programming are addressed in the science and engineering practices of the Next Generation Science Standards. In other words, both the Standards address the fluency with which coding can be integrated into science and mathematics. In accordance with the reporting of the informants, Weavers: design constructivist activities and environments for learning; focus on forming and transforming student identity; identify as self-advocates; employ programming languages that emphasize computation; are active at conferences; and assess code as a process of learning.

Learning environment. Weavers described coding activities where students are engaged in hands-on learning as they *play around* with variables in order to make connections between individual components of a system and the larger system. Programming provides immediate feedback for students as they change a variable or alter a portion of code and run the program again. In addition, the feedback they receive is self-initiated and based purely on their actions, meaning that the feedback is individualized and meaningful. Interestingly, each of the Weavers described themselves as instructionists normally; however, the practice of recontextualizing code naturally transformed each of them to a more constructivist approach in the classroom. Not only did their practice change, but Weavers described themselves as facilitators during recontextualization activities, thereby addressing an identity shift within themselves. It is not clear whether coding lends itself to constructivism or whether the informants got ideas from other sources, and the activities happened to be constructivist. Nonetheless, shifting from an instructionist style to a constructivist practice was a phenomenon among all Weavers.

Student identity. Weavers perceived student identity to be a barrier for learning science; that is, underrepresentation in many scientific fields has created a culture where students do not feel they belong. Weavers, then, created constructivist activities where students behave as scientists as they hypothesize, test, and revise their initial thinking to come up with new models. Both S1 and S8 made this connection visible for students, while S3 focuses on pointing out role models that may appeal to his students. It is interesting to note that science (excluding biology), math, and computer science suffer from a lack of women and people of color, and yet computer science is being used to increase students' perception of self as a practitioner within science and math. More information is needed to determine if computer science implemented as a method of constructivism within math and science classes changes students' perception of self, and why coding may have that effect.

Self-advocacy. Unlike Communicators, the Weavers were not trained to implement code from local professional development sessions. Both S3 and S8 worked in the IT industry before their career as a teacher, and S1 learned computer science in college as a science major. Interestingly, all three of them hold a Master's degree in education, and S8 specifically noted that he was introduced to programming in education in his graduate program. All of the Weavers reported that they often attend and present at local and national educational conferences. Clearly, the Weavers are self-advocates, and were highly motivated to make change in their classrooms.

Programming languages. The Weavers chose programming languages designed specifically for computation, as the practice is centered on data manipulation and analysis. Informants reported that Microsoft Excel and Python were used for simulations, while the

Sphero was used to convert speed and time into distance. The informants who used Microsoft Excel and Python noted that teaching the basics of the technology was a challenge; their focus was to engage and increase students' attitudes toward science, and they felt that there was no engaging method of introducing coding basics. Unlike Scratch, Python and Excel require some initial knowledge before students can apply code creatively to their own projects. For math and science teachers to become Weavers, they will need to figure out how to provide engaging activities for students to become proficient enough with code to creatively integrate it into their projects.

Assessment. Each of the Weavers described his/her focus on students' process over product. The informants described their usual assessment outside of recontextualization as based on student reasoning. Because Weavers integrate coding into their content standards, assessment of code was similar. Code was not explicitly assessed; rather, students described how code influenced their process, and the activity was graded according to their normal schema.

The Cultural-Historical Activity of Weavers Recontextualizing Code

Like the Communicators, the Weavers were successful in their recontextualization of code (a criterion for participation in this study). Weavers use computer science as a vehicle for students to explore data using student-centered, hands-on, constructivist approaches. Teachers generally come from math and science backgrounds, where they have seen first-hand the usefulness of programming within their respective field. Weavers focus on forming and transforming student identity as an intervention for underrepresentation within certain STEM fields. Figure 12 shows the diagram of a Weaver's system as they engage in recontextualizing code. According to sociocultural activity theory, a Weaver's practice is mediated by tools and symbols, the rules of the system, community members, and the division of labor.

Tools and symbols. Weavers chose programming languages and environments that were designed specifically for computing such as Microsoft Excel and Python. Weavers use data manipulation and analysis as the basis of their recontextualization of code for the purpose of mathematical or representational modeling. During recontextualization activities, students use the immediate feedback feature of programming to test and revise their simulations. Consequently, computational programming languages such as Excel and Python are used to engage students in hands-on learning experiences and through that experience form and transform students' identities toward that of scientists. At least, that is the goal for Weavers. Unfortunately, some tensions exist within the system that create barriers for this goal. For example, Weavers described the difficulty of teaching enough of the programming language so that students felt comfortable to explore and play during the project. Weavers noted that students felt the initial learning of Python or Excel to be boring; however, they did not have an answer for how to teach the basics more enjoyably. Because of this, many students were turned off to coding from the get-go, and never experienced the simulation as designed. It is worth noting that the programming language used by S4 is similar to Scratch, and she did not experience difficulty in teaching the language because of its low floor. It is worth investigating how Weavers use programming languages like Scratch.

Rules. S1 told the story of how she moved to Texas from Puerto Rico, where she had been recontextualizing code for years. At a public school in Texas, S1 was told by her administrator that she could not have computers, and that she was not allowed to use coding in her class because coding does not increase scores on the state exams. S1's administration was concerned that S1 would not complete all of the standards and that her students would bring down the school's scores. S1 said that despite how much she tried to explain that recontextualization would actually increase student understanding, her administration did not understand because the administrator had no background in science. To move beyond the tension in her system, S1 moved to a private school where she would be able to recontextualize code. It is interesting to note that the administrator at the public school in Texas did not respect S1 as a professional, even without knowledge of science education.

Community. Weavers perceived themselves to be highly active at local conferences. They perceived themselves to be part of local communities that share knowledge, and continually add to that knowledge over time in order to find best practices for engaging students and increasing student understanding. Despite their participation at conferences, Weavers noted that they feel isolated at their school site, and do not have other colleagues with which to trade ideas. Weavers then turned to Twitter and other forms of social media to seek communities online of like-minded individuals within their respective fields who might support their practice and help reflect on current practices.

Division of labor. Despite feeling part of a community of conference-attendees as well as an online community, Weavers described their practice as isolationist, where they create all of their materials, find their own ways to support students in the classroom, develop their own assessments, and reflect on their own. Although they might get ideas for projects at conferences and others online, Weavers described the need to test the project, and make adjustments over time on their own. Due to the isolationist manner in which Weavers work, it makes sense that they find difficulty in teaching the basics of coding when there is significant literature about equitable practices and inquiry methods of teaching code. Unfortunately, an organized group for Weavers does not currently exist, and since Weavers are self-advocates, they will find any method of seeking information related to their goals in the classroom.

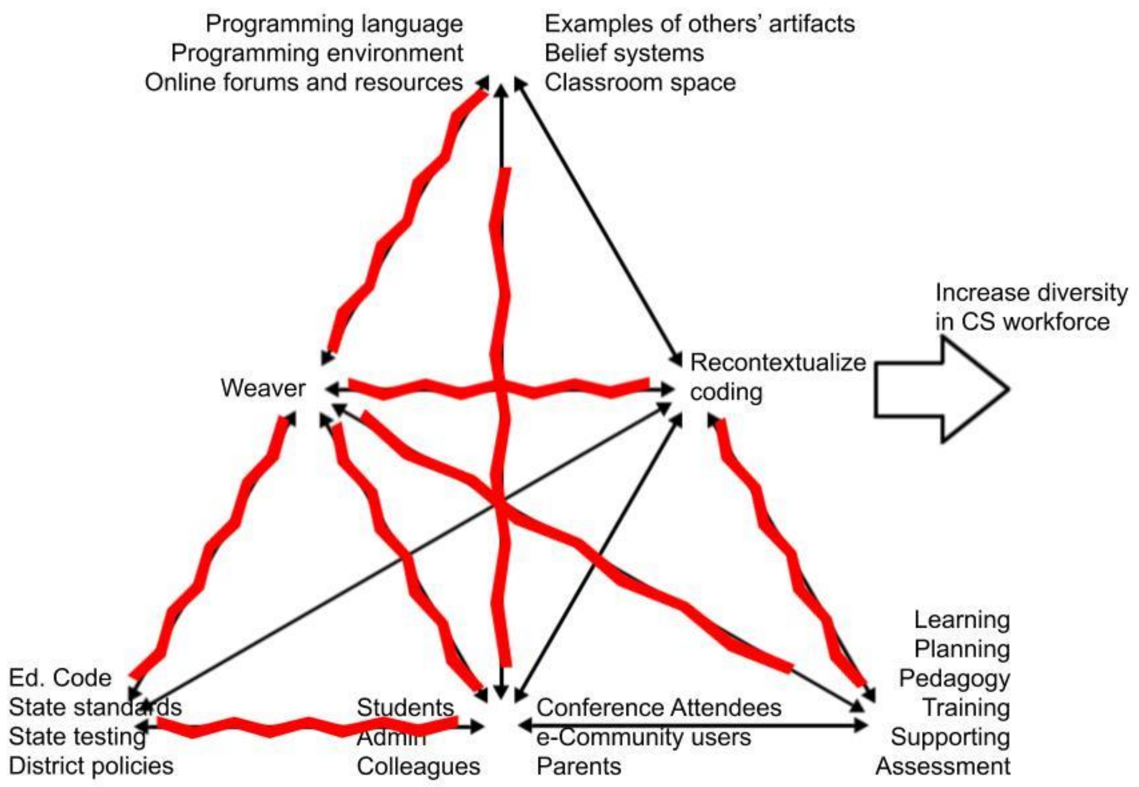


Figure 12. Cultural-historical activity of Weaver recontextualizing code.

Implications for Schools of Education

Schools of education are responsible for preparing pre-service teachers for the challenges of teaching within the K-12 educational system. Course of study generally includes child development and psychology, general curriculum and lesson design, handling diversity within a multiethnic classroom, supporting special populations such as English language learners and students with special needs, integrating computers in instruction, and a methods course specifically related to teachers' content area. It is important to note that as of 2019, computational thinking and computer programming are rarely part of a traditional pre-service program, and in those rare cases, are integrated into a Methods of Teaching Science course due to computational thinking's placement in the Next Generation Science Standards. The data suggests that schools of education should begin to incorporate computational thinking and programming into their curricula for all teachers, not just those whose standards explicitly reference them.

Communicator Training and Support

Each of the Communicators was trained by private organizations, individuals at conferences, or Master's programs for educational technology, and were further supported at his/her respective site. One of the Communicators, S6, was told by her administrators that they wanted to see more technology used in her class, and she described being intimidated by the prospect of incorporating robotics into her English class, as she does not perceive herself to be confident with technology. All teachers at her school site were trained by representatives from BirdBrain Technologies, and S6 was personally supported by a mentor teacher at her site who had a strong relationship with BirdBrain Technologies. Another Communicator, S2, described feeling isolated in her rural valley in California where trainings are scarce. She described taking the initiative to seek others on Twitter who were recontextualizing code, and found a community of practitioners who supported her development and identity as a teacher who uses code.

Recontextualizing code is a new development in education, and organizational structures do not exist for training and supporting teachers. In fact, computer science credentials only exist in certain locations, and training programs for computer science teachers in general are dependent on individuals, and are locally available within regions where individuals are working to broaden participation in computer science education. Nonetheless, the activity described by the Communicators redefines what it means to do computer science, and is a more general activity that can be integrated by any teacher, not just computer science teachers. In other words, the Communicators described coding as an activity that all teachers can implement to engage students in communicating their learning. Recontextualizing code then transcends the educational tradition of learning variables, conditionals, loops, and algorithms without context, as well as contextualizing code, where diverse content is integrated into a computing class in order to contextualize the learning of variables, conditionals, looping, and algorithms. The Communicators recontextualized code from within their own domains, and treated code like any other communication technology.

Communicators and Media Literacy

Each of the Communicators expressed a desire for all students to code. Their choice of programming language and environment alleviated the need to spend time learning code before the projects so that students could learn while building. Integrating code as an encapsulating environment not only situates and contextualizes activity within a project-based learning environment, but also provides space for students to be creative and personalize their constructed media. This type of argument has been made before in media literacy: in the 1990s, students were supposed to be literate in Hypercard and presentation software; and in the 2000s, it was video creation and web design. It could be argued that coding as a literacy is not learning from the past.

The author would argue in turn that the world was not yet ready for the skills learned in past literacy movements to have meaning for our students. For example, streaming on YouTube and Twitch is now a major industry, and video creation and curation is a skill that many of our current students who are or want to be streamers must possess. It could be argued that the YouTube and Twitch of coding has not yet been created, and once the technology and industry catch up with the literacy, then students will find great meaning in coding. It even could be argued that we should be incentivizing others to design the technologies and structures to begin this newfound industry, where amateur coders can make meaningful content within a global market.

Consider the economic, social, and cultural impact if coding became a literacy across the United States. Technology has vastly changed the landscape of human experience; however, everyone is currently still a user rather than a producer of coded content. What might the world look like if instead of users, everyone became a coder? If coding is to become a literacy, it must be inserted into education systems broadly and generally. The Communicators described scenarios where students required limited knowledge about code before entering projects, the programming language and environment allowed for novices and experts to be engaged, and students created media to communicate their learning of content; that is, any teacher can implement Communicator code in any class. The Communicators described a style of implementing code in the classroom that can be scaled broadly and for any classroom. All that is required to initiate a code literacy movement is a sustainable model for teacher preparation.

Most states like California have a technology requirement for preliminary credentials. Educational technology classes within schools of education fulfill states' technology requirement, and are designed to provide pre-service teachers with strategies and skills to prepare their students to be lifelong learners in an information-based society. If coding is included as a lifelong skill required for the future, then schools of education should present coding as a Communicator skill within the technology portion of pre-service curricula. This would require conversations about content that arose from the informants: transitioning toward a co-learning model of teaching rather than presenting information; managing the chaos of groups working on different tasks; managing technology and planning for technological failure; and supporting students in their endeavors.

Communicators and the Masculine Occupational Culture of Computing

The Communicators benefited from working with those within educational spaces rather than those in the IT industry. S2 described the chilly climate resulting from attempting to partner with the computer science teacher on her campus. The masculine occupation of computing gendered the activity of coding, which S8 cited as the reason women, students of color, and intersections thereof historically have not participated in computer science classes. As coding becomes an activity that all teachers can engage in for the purpose of Communicating, it is important to keep the activity of educational coding distinct from the practices and cultures inherent within industry.

Weavers and Content

Distinct from the Communicators, the Weavers were all math and science teachers, and weaved coding into their content standards. For example, S1 and S6 reported to use simulations in Excel and Python to help students play with variables in their inquiries into systems, S3 had students create short computational programs in Python to check their Chemistry and math homework, and S4 reported to use a Sphero to bridge content in Geometry and Physics. Each of the activities that teachers developed revolve around data manipulation and drawing conclusions from data; that is, simulations are used to determine how changes in data values yield changes in the larger system, computational programs like S3 devised manipulate data using mathematical operations, and S4's use of the Sphero converted physics data into geometric data. Data storage, data manipulation, and data access are key functions of programming languages, and serve as an intersection of math, science, and computer science. Interestingly, all four of the STEM teachers who used code to mediate course content did so specifically regarding data manipulation and storage.

Schools of education should introduce Weaving as a cross-curricular support for student inquiry into Methods courses for math and science teachers, which required significant support for teachers. The manner in which Weavers reported to use data storage and manipulation as the intersectional space of math, science, and computer science requires significantly more preparation for students than Communicators experienced. Unlike the Communicators, who chose Scratch in order to reduce the amount of instruction before projects, S1, S3 and S8 used computational languages like Excel and Python, which require significant instruction in how to use the programming environment as well as how to code. For example, each Weaver described methods of introducing code to students and supporting students in learning basic coding. S8 lamented that he does not have an engaging, constructivist method for introducing how to create functions and simulations in Microsoft Excel, and reported to continue to search for student-centered models for teaching coding and the programming environment. S3 gave students model code from which they were meant to extrapolate their own code. Weavers were challenged by the need to support all learners in the basics of coding, but also not spending too much time on projects so that they could cover every standard in preparation for state exams. S3 admitted that not all students enjoyed coding in his class, and this is unfortunate, as negative experiences will most likely dissuade those students from future coding opportunities (Dewey, 1986). Methods courses in math and science at colleges of education should focus on pedagogies of coding that emphasize equitable practices to give all students a positive experience.

General Recommendations for Schools of Education

Both Communicators and Weavers described receiving continued support from teachers on their school site or online communities. Many colleges of education provide face-to-face professional development opportunities for teachers in the local community as well as online, synchronous sessions. Schools of education should collaborate with computer science departments to provide continued professional development and support for teachers who use code in the classroom, including by recontextualizing. Those who provide community support should be mindful of the masculine culture of computing and purposefully insert structures that reduce chilly climates.

Implications for colleges of education are dependent on the relationship between code and teachers' content standards, as this has been shown to dictate the type of activities given to students, how students are assessed, and the types of support and training required for teachers. Due to the lack of a relationship between coding and content for Communicators, their activity can be scaled across all disciplines. Student activity is embedded within developing a narrative of their learning, and because Communicating is a style of project-based learning, assessment and support are part of the project itself. Schools of education can embed coding as a Communicator activity within any class that supports student engagement and use of technology; whereas Weavers are more specialized and purposeful in their use of code. Quantitative data manipulation and analysis is embedded within mathematics and science standards, and Weavers reported to use coding as a means of developing inquiry activities for students to play with data. Colleges of education should provide instructional strategies and modeling for math and science teachers in specific Methods courses.

Implications for Students

Although students were not interviewed for this study, informants described their perceptions of the impact of recontexualizing code on students. All teachers reported an increase in engagement during coding projects. There is power in coding; technology is not neutral (see Chapter Two). There is power in providing students with the ability to design their own projects based on the needs of their communities. Currently, communities of underrepresented groups are not included in the development and testing of new technologies; recontextualization is a first step in a future where anybody can design a technological intervention for their community. Due to the differences in the activity systems of Communicators and Weavers, their perceptions of student impact also differed.

Communicators' perceptions of student impact. Many Communicators described students coming into class during nutrition or lunch to work on coding projects outside of school such as making others aware of local animal shelters. Using coding as a means of informing others in the community aligns with the Communicator practice of coding as a means of communication and self-expression. Communicator recontexualization supports the notion of coding as a media literacy in that students use code as a vehicle for media creation. Additionally, Communicators described students supporting their peers during coding projects. Interestingly, this was an experience that Weavers described. A few of the Communicators reported some gender politics influencing student behavior and attitude toward coding; although all Communicators noted that there is no difference in abilities between men and women, they acknowledged that oftentimes, men are more confident in their coding than women. Unfortunately, the masculine culture of computing has been internalized by students, creating a barrier for women and students of color even when code is recontextualized by Communicators.

Weavers' perceptions of student impact. One of the tensions described by Weavers was the inability to initially teach enough of Python and Microsoft Excel while still keeping students engaged and motivated to learn. Interestingly, the Weaver who used the Sphero did not experience the same tension as the teachers who used simulations. In fact, one of the benefits of Scratch, as described by Communicators, was its low floor and lack of a need for prefacing projects with coding lessons. Although teachers who ran simulations used programming languages designed specifically for computations, it would be interesting to observe a Weaver running simulations using Scratch. Further research must be done on future Weavers who use low-floor languages and environments, and how the modification in technology affects students as well as teacher practice.

Issues Arising from This Study

During the process of finding subjects for this study, some issues arose regarding current computer science educational research and practice. The author found that many teachers have differing beliefs about the definition of computational thinking, and there was one computer science teacher who attempted to sabotage the study.

Computational thinking. One potential subject wrote in a preliminary email that she uses software to teach an elementary level bilingual math course, and the software engages students in computational thinking. During the interview, the researcher asked in what ways the software does this. She responded, “It's like looking at patterns and then finding out what to do to solve the math problem.” The researcher pressed her on how solving math problems and looking for patterns is computational thinking, and she replied, “So it's more like looking at patterns or just figuring out what to do with the game. So it's sort of a game of feeling where they have to figure out what to do.” It was obvious that the teacher could not communicate how the program used computational thinking, and it turned out that at the beginning of the year, she had been trained for an hour on how to maneuver through the program, and was told plainly by the trainers that it uses computational thinking. It can sometimes be difficult to distinguish thinking in a mathematics class from computational thinking, and curriculum developers and program spokespeople can oftentimes obfuscate the two. Another potential informant never had an interview because when asked how he employs computational thinking in his math class, he simply said that math is computation, and students have to think about their computations—thus, computational thinking.

Masculine culture of computing. N2 was found in a Facebook forum that one of the initial PIs suggested would be good. The interview with N2 began, and he directed the researcher to his website where he has some very impressive projects made by students. While doing this, he said, "I understand that you were really more interested in non-programming courses and how they use computation, but that kind of describes my programming course because my programming course is contextualized inside academic subjects outside of computer science." The researcher explained that N2's class contextualizes coding, but his class is inherently a coding class, and does not fit the criteria of the study. N2's voice became aggravated, and he asked, "So my question to you then is what is the programming they're learning? Really, it's trivialized because programming takes a lot of time to learn. Programming is a skill." He then continued:

> You really have to change the pedagogy, change the instruction, change instructional strategies to make it look more like a language course, which is a lot of repetition and a lot of practice rather than zooming through things. And you have to stay with the same language for a couple of years until kids reach proficiency. So this is it. The thing what strikes me about. I mean, you can have kids, you know, create things in scratch pretty quickly and pretty easily. But that doesn't mean they know how to program at all. It just means they're building something, you know, they're dragging and dropping something and they're not going to really learn how to program.

N2 made an argument that has been heard before; that is, ‘Scratch is not real programming. Real programming by real programmers takes years to learn, and it is specialized and specific and this is definitely not it.’ Immediately following this rant, the researcher thanked N2 for his time and ended the interview. The following day, the researcher received three emails from math and science teachers who had commented in the same Facebook group as N2 was found, and whose interviews were already planned. Each said that N2 had sent them a personal message about the study, and they did not feel comfortable participating. One person even said that the researcher should be ashamed for wanting to take jobs away from hard-working computer science teachers.

Next Steps in Research and Practice

This was an exploratory, phenomenological study to understand the experiences and perspectives of teachers who recontextualize code. The study found that there are two types of recontextualizers, Communicators and Weavers, and each type has different activity systems that mediate their practice. Considering the importance of coding on future human activity and the need to increase diversity among those who create future technologies, recontextualizing code is an example of a method of delivering computer science content to all students, and deserves both further study and a scaled implementation across school districts and states. Developing Communicators and Weavers at a large scale within the current educational system includes both policy changes at multiple levels, the construction of networks and pathways to support both students and teachers, and the integration of research and practitioner communities.

Future research for Communicators. One of the barriers for potential Communicators is administrator’s focus on state standards, and the belief that project-based learning wastes valuable time and does not increase scores. If more schools are to support Communicator recontextualization, then research regarding the efficacy of recontextualization relative to student understanding of content must be conducted. Beyond that initial research, it would be worthwhile to determine whether Communicator recontextualization increases the chances of students using computer science in their future endeavors or taking future computer science courses. A longitudinal study would be required to determine whether students continue to use coding meaningfully in their everyday endeavors or whether students choose to take further computer science classes if given the opportunity. During that study, it would also be interesting to determine how students use code in their everyday endeavors beyond their experience in a course where coding was recontextualized.

Future research for Weavers. In this study, all of the Weavers who ran simulations in the classroom used Microsoft Excel or Python. Although Excel and Python are designed for computation, Weavers described the challenge of introducing sufficient coding to enable students to test and play with their simulations without becoming disenfranchised. This experience is in direct opposition of the Communicators, who chose Scratch specifically for its usability and low entrance barrier and high ceiling. The next study for Weavers would look at Weaver's practice using a programming language like Scratch, and analyze student perception of self as a mathematician or scientist and attitude toward computer science, since those were the constructs that Weavers focused their attention. Because Weavers use programming as a means of mediating student understanding of content standards, the following quasi experimental study would analyze two different constructivist classrooms, where one recontextualizes code, and determine the efficacy of recontextualizing code in a math or science classroom. Follow-up studies would inspect the use of peer pedagogy, problem solving techniques of students, and teachers' equitable and inquiry practices during recontextualization activities.

Policy. In order to increase the number of Communicators and Weavers in the United States, scalable professional development opportunities must be created specifically with the goal of training teachers in choosing appropriate programming languages and environments, focusing on equity and access, providing best practices, and a community of continued support. Research would need to be conducted on participant experience during Communicator and Weaver professional development in order to determine whether teachers recontextualize code beyond a professional development as well as their belief systems regarding student achievement, the value of recontextualization. That study would also survey students in classes of teachers who exited a professional development in order to determine student attitude toward computer science as well as perceptions of identity.

Micro credentials. Micro credentials should be developed much like the recent CLAD credential, which is a measure of a teacher’s training in supporting English language learners. The California computer science state standards were designed specifically to be integrated into general classes as well as stand-alone computer science classes, and can be used as a basis for curriculum and assessment for the micro credentials. Two different micro credentials should be developed: one for future Communicators, and one for future Weavers. Like the CLAD, micro credentials for Communicators and Weavers can be integrated into pre-existing course work within credential programs with some modifications. Once the micro credentials are created, studies regarding knowledge sharing across spaces, communities of practice, and the mediation of teacher identities across multiple spaces could be conducted.

State-wide and district policy change. After micro credentials are developed, states and districts should enact policies incentivizing the development and implementation of local professional developments for technology use and computer science in the classroom. It was not coincidence that three of the Communicators in this study were from Pennsylvania; Pennsylvania has enacted a multi-layered effort to increase STEM learning across the state. It was not easy finding subjects for this study, and for three Communicators to come from Pennsylvania is a testament to how providing free, accessible, and meaningful professional development can have an impact on teachers' individual practices. Considering the benefits of training an entire generation of youth to become code developers and content creators, it is worthwhile to explore state and district policy changes, even if those policy changes enforce the use of micro credentials.

REFERENCES

Alba, D. (2015, January 23). Finally, you'll be able to track your period in iOS. *Wired*. Retrieved from https://www.wired.com/2015/09/finally-youll-able-track-period-ios/

Anderson, N. S., Norman, D. A., & Draper, S. W. (1988). User centered system design: New perspectives on human-computer interaction. *The American Journal of Psychology*, *101*(1), 148. https://doi.org/10.2307/1422802

AP program participation and performance data 2018. (2018). Retrieved from https://research.collegeboard.org/programs/ap/data/participation/ap-2018

Archer, G., Bohmann, L., Carter, A., Cischke, C., Ott, L. M., & Ureel, L. (2016). Understanding similarities and differences in students across first-year computing majors. *Proceedings - Frontiers in Education Conference, FIE*, *November*, 1–8. https://doi.org/10.1109/FIE.2016.7757695

Ashcraft, C., & Blithe, S. (2010). *Women in IT: The facts*. Retrieved from www.ncwit.org

Ashcraft, C., McLain, B., & Eger, E. (2016). *Women in tech: The facts*. Retrieved from https://www.ncwit.org/resources/women-tech-facts-2016-update

Aspray, W. (2016). *Participation in computing: The National Science Foundation's expansionary programs*. https://doi.org/10.1007/978-3-319-24832-5

Astrachan, O., Gray, J., Beth, B., Osborne, R. B., & Lee, I. (2014). Diverse learners, diverse courses, diverse projects. In J. Dougherty & K. Nagel (Chairs), *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 177–178). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/2538862.2538991

Bardram, J. E. (1997). Plans as situated action: An activity theory approach to workflow systems. In J. Hughes, W. Prinz, T. Rodden, & K. Schmidt (Eds.), *Proceedings of the Fifth*

European Conference on Computer Support Cooperative Work (pp. 17–32). Lancaster, UK: Springer. https://doi.org/10.1007/978-94-015-7372-6_2

Barr, V., & Stephenson, C. (2012). Bringing computational thinking to K-12. *ACM Inroads*, *2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Bartol, K. M., & Aspray, W. (2006). The transition of women from the academic world to the IT workplace: A review of the relevant research. In J. McGrath-Cohoon & W. Aspray (Eds.), *Women and Information Technology: Research on Underrepresentation* (pp. 378–419). https://doi.org/10.7551/mitpress/9780262033459.003.0013

Beyer, S., Rynes, K., & Haller, S. (2004). Deterrents to women taking computer science courses. *IEEE Technology and Society Magazine*, *23*(1), 21–28. https://doi.org/10.1109/MTAS.2004.1273468

Blackler, F. (2009). Cultural-historical activity theory and organization studies. In A. Sannino, H. Daniels, & K. D. Gutierrez (Eds.), *Learning and Expanding with Activity Theory* (pp. 19–39). https://doi.org/10.1017/CBO9780511809989.003

Blikstein, P. (2013). Digital fabrication and 'making' in education: The democratization of invention. In J. Walter-Herrmann & C. Buching (Eds.), *FabLabs: Of Machines, Makers and Inventors* (pp. 1–21). https://doi.org/10.1080/10749039.2014.939762

Blunden, A. (2010). *An interdisciplinary theory of activity*. https://doi.org/10.1163/ej.9789004184060.i-344

Bødker, S. (1997). Computers in mediated human activity. *Mind, Culture, and Activity*, *4*(3), 149–158. https://doi.org/10.1207/s15327884mca0403_2

Bonar, J., Ehrlich, K., Soloway, E., & Rubin, E. (1982). Collecting and analyzing on-line protocols from novice programmers. *Behavior Research Methods & Instrumentation*, *14*(2), 203–209. https://doi.org/10.3758/BF03202154

Bowers, C. A. (1997). The paradox of technology: What's gained and lost? *The NEA Higher Education Journal*, *14*(1), 49–58. Retrieved from http://beta.nsea-nv.org/assets/img/PubThoughtAndAction/TAA_00Fal_12.pdf

Breland, A. (2017, December 4). How white engineers built racist code – and why it's dangerous for black people. *The Guardian*. Retrieved from https://www.theguardian.com/technology/2017/dec/04/racist-facial-recognition-white-coders-black-people-police

Brown, P., & Concannon, J. (2018). Next generation science standards (NGSS). In P. Brown & J. Concannon (Eds.), *Inquiry-Based Science Activities in Grades 6–12* (pp. 9–18). https://doi.org/10.4324/9781351064583-2

Bruner, J. S. (1956). *A study of thinking*. New York, NY: Wiley.

Buckingham, D. (2003). *Media education: Literacy, learning and contemporary culture*. https://doi.org/10.1002/9781444344158

Buechley, L., Eisenberg, M., Catchen, J., & Crockett, A. (2008). The LilyPad Arduino. In M. Czerwinski & A. Lund (Chairs), *Proceeding of the Twenty-Sixth Annual CHI Conference on Human Factors in Computing Systems - CHI '08* (pp. 423-432). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/1357054.1357123

California State Board of Education. (2018). *Computer Science Education - Content Standards*. Retrieved from https://www.cde.ca.gov/be/st/ss/computerscicontentstds.asp

Carr, P. L., Helitzer, D., Freund, K., Westring, A., McGee, R., Campbell, P. B., … Villablanca, A. (2018). A summary report from the research partnership on women in science careers. *Journal of General Internal Medicine*, *34*(3), 356-362. https://doi.org/10.1007/s11606-018-

4547-y

Caukin, N., & Trail, L. (2019). SAMR: A tool for reflection for Ed Tech integration. *International Journal of the Whole Child*, *4*(1), 47–54. Retrieved from https://libjournals.mtsu.edu/index.php/ijwc/article/view/1370

Chemaly, S. (2016, March 16). The problem with a technology revolution designed primarily for men. *Quartz*. Retrieved from https://qz.com/640302/why-is-so-much-of-our-new-technology-designed-primarily-for-men/

Cheryan, S., Plaut, V. C., Davies, P. G., & Steele, C. M. (2009). Ambient belonging: How stereotypical cues impact gender participation in computer science. *Journal of Personality and Social Psychology*, *97*(6), 1045–1060. https://doi.org/10.1038/jhg.2010.44

Cooper, S., Pérez, L. C., & Rainey, D. (2010). K--12 computational learning. *Communications of the ACM*, *53*(11), 27–29. https://doi.org/10.1145/1839676.1839686

Creswell, J. W. (2018). *Research design: Qualitative, quantitative, and mixed methods approaches*. Los Angeles, CA: SAGE Publications.

Cross, F. (2017). *Teacher shortage areas nationwide listing: 1990-1991 through 2016-2017*. https://doi.org/10.1177/1362168812455588

CS Unplugged. (2019). *Binary candles or normal candles on your cake?* Retrieved from https://csunplugged.org/en/topics/binary-numbers/binary-or-normal-candles/

Cuéllar, M. P., & Pegalajar, M. C. (2014). Design and implementation of intelligent systems with LEGO Mindstorms for undergraduate computer engineers. *Computer Applications in Engineering Education*, *22*(1), 153–166. https://doi.org/10.1002/cae.20541

Cuny, J. (2015). Transforming K-12 computing education. *ACM Inroads*, *6*(3), 54–57. https://doi.org/10.1145/2809795

Cuny, J., & Jan. (2015). Transforming K-12 computing education. *ACM Inroads, 6*(4), 58–59. https://doi.org/10.1145/2832916

Dahlbom, B., & Mathiassen, L. (1997). The future of our profession. *Communications of the ACM*, *40*(6), 80–89. https://doi.org/10.1145/255656.255706

Davies, P. G., Spencer, S. J., Quinn, D. M., & Gerhardstein, R. (2002). Consuming images: How television commercials that elicit stereotype threat can restrain women academically and professionally. *Personality and Social Psychology Bulletin*, *28*(12), 1615–1628. https://doi.org/10.1177/014616702237644

Davis, E. A., Linn, M. C., & Clancy, M. (1995). Learning to use parentheses and quotes in LISP. *Computer Science Education*, *6*(1), 15–31. https://doi.org/10.1080/0899340950060102

Deal, T. E., & Kennedy, A. A. (2000). *Corporate cultures: The rites and rituals of corporate life*. Cambridge, MA: Perseus Books.

Dewey, J. (1986). Experience and education. *Educational Forum*, *50*(3), 242–252. https://doi.org/10.1080/00131728609335764

DiCicco-Bloom, B., & Crabtree, B. F. (2006). The qualitative research interview. *Medical Education*, *40*(4), 314–321. https://doi.org/10.1111/j.1365-2929.2006.02418.x

DiSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

Drew, N. (2004). Creating a synthesis of intentionality: The role of the bracketing facilitator. *Advances in Nursing Science*, *27*(3), 215–223. https://doi.org/10.1097/00012272-200407000-00006

Easter, B. (2018). "Feminist_brevity_in_light_of_masculine_long-windedness:" Code, space, and online misogyny. *Feminist Media Studies, 18*(4), 675–685.

https://doi.org/10.1080/14680777.2018.1447335

Eccles, J. S. (1987). Gender roles and women's achievement-related decisions. *Psychology of Women Quarterly*, *11*(2), 135–172. https://doi.org/10.1111/j.1471-6402.1987.tb00781.x

Eglash, R., Bennett, A., O'Donnell, C., Jennings, S., & Cintorino, M. (2006). Culturally situated design tools: Ethnocomputing from field site to classroom. *American Anthropologist*, *108*(2), 347–362. https://doi.org/10.1525/aa.2006.108.2.347

Ellis, A., & Heneghan, L. (2016). *Harvey Nash / KPMG CIO Survey 2016*. Retrieved from www.hnkpmgciosurvey.com

Engeström, Y. (1987). *Learning by expanding: An activity theoretical approach to developmental research*. Cambridge, MA: Cambridge University Press.

Engeström, Y. (2000). Activity theory as a framework for analyzing and redesigning work. *Ergonomics*, *43*(7), 960–974. https://doi.org/10.1080/001401300409143

Engeström, Y. (2001). Expansive learning at work: Toward an activity theoretical reconceptualization. *Journal of Education and Work*, *14*(1), 133–156. https://doi.org/10.1080/13639080123238

Engeström, Y. (2011). Activity theory and learning at work. In M. Malloch, L. Cairns, & B. O'Connor (Eds.), *The SAGE Handbook of Workplace Learning* (pp. 86–104). https://doi.org/10.4135/9781446200940.n7

Engeström, Y., Miettinen, R., & Punamäki, R. (1999). *Perspectives on activity theory*. Cambridge, MA: Cambridge University Press.

Ensmenger, N. L. (2001). The "question of professionalism" in the computer fields. *IEEE Annals of the History of Computing*, *23*(4), 56–74. https://doi.org/10.1109/85.969964

Ensmenger, N. L. (2010). Making programming masculine. In T. J. Misa (Ed.), *Gender Codes:*

Why Women Are Leaving Computing (pp. 115–141). Retrieved from https://s3.amazonaws.com/academia.edu.documents/31618402/Ensmenger2010-MPM.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1533494354&Signature=I%2FgNJkFHrjvCBqqtqbUdSjFqOAc%3D&response-content-disposition=inline%3B filename%3DMaking_programming_masculine

Ensmenger, N. L. (2012). *The computer boys take over: Computers, programmers, and the politics of technical expertise*. Cambridge, MA: MIT Press.

Faulkner, W. (2001). The technology question in feminism: A view from feminist technology studies. *Women's Studies International Forum*, *24*(1), 79–95. https://doi.org/10.1016/S0277-5395(00)00166-7

Faulkner, W. (2009). Doing gender in engineering workplace cultures. *Engineering Studies*, *1*(1), 3–18. https://doi.org/10.1080/19378620902721322

Feurzeig, W., Papert, S., & Lawler, B. (1969). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5), 487--501. https://doi.org/10.1080/10494820903520040

Fields, D. A., Kafai, Y. B., & Giang, M. T. (2016). Coding by choice: A transitional analysis of social participation patterns and programming contributions in the online Scratch community. In U. Cress, J. Moskaliuk, & H. Jeong (Eds.), *Mass Collaboration and Education* (pp. 209–240). https://doi.org/10.1007/978-3-319-13536-6_11

Fields, D. A., Lui, D., & Kafai, Y. B. (2017a). Teaching computational thinking with electronic textiles: High school teachers' contextualizing in Exploring Computer Science. In S. Kong, J. Sheldon, & K. Y. Li (Eds.), *Conference Proceedings of International Conference on Computational Thinking Education* (pp. 67–72). Hong Kong: The Education University of

Hong Kong.

Fields, D. A., Lui, D., & Kafai, Y. B. (2017b). Teaching computational thinking with electronic textiles: High school teachers' contextualizing in Exploring Computer Science. In S. Kong, J. Sheldon, & K. Y. Li (Eds.), *Conference Proceedings of International Conference on Computational Thinking Education* (pp. 67–72). Retrieved from http://www.eduhk.hk/cte2017/doc/CTE2017 Proceedings.pdf#page=78

Fields, D. A., Vasudevan, V., & Kafai, Y. B. (2015). The programmers' collective: Fostering participatory culture by making music videos in a high school Scratch coding workshop. *Interactive Learning Environments*, *23*(5), 613–633. https://doi.org/10.1080/10494820.2015.1065892

Fisher, A., & Margolis, J. (2002). Unlocking the clubhouse: The Carnegie Mellon experience. *SIGCSE Bull.*, *34*(2), 79–83. https://doi.org/10.1145/543812.543836

Fisher, A., & Margolis, J. (2003). Unlocking the clubhouse. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education - SIGCSE '03*, *34*(2), 79–83. https://doi.org/10.1145/611892.611896

Fitzsimons, A. (2002). *Gender as a verb: Gender segregation at work.* New York, NY: Routledge.

Form, D., & Lewitter, F. (2011). Ten simple rules for teaching bioinformatics at the high school level. *PLoS Computational Biology*, *7*(10), e1002243. https://doi.org/10.1371/journal.pcbi.1002243

Frehill, L. M. (2004). The gendered construction of the engineering profession in the United States, 1893–1920. *Men and Masculinities*, *6*(4), 383–403. https://doi.org/10.1177/1097184X03260963

Gadanidis, G. (2015). Coding as a Trojan horse for mathematics education reform. *Journal of Computers in Mathematics and Science Teaching*, *34*(2), 155–173. Retrieved from https://eric.ed.gov/?id=EJ1061256

Goles, T., Hawk, S., & Kaiser, K. M. (2009). Information technology workforce skills: The software and IT services provider perspective. *Information Systems Outsourcing (Third Edition): Enduring Themes, Global Challenges, and Process Opportunities*, *10*(2), 105–125. https://doi.org/10.1007/978-3-540-88851-2_5

Goode, J. (2007a). If you build teachers, will students come? The role of teachers in broadening computer science learning for urban youth. *Journal of Educational Computing Research*, *36*(1), 65–88. https://doi.org/10.2190/2102-5G77-QL77-5506

Goode, J. (2007b). If you build teachers, will students come? The role of teachers in broadening computer science learning for urban youth. *Journal of Educational Computing Research*, *36*(1), 65–88. https://doi.org/10.2190/2102-5G77-QL77-5506

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: The exploring computer science program. *ACM Inroads*, *3*(2), 47–53. https://doi.org/10.1145/2189835.2189851

Goode, J., Estrella, R., & Margolis, J. (2006). Lost in translation: Gender and high school computer science. In J. Cohoon & W. Aspray (Eds.), *Women and Information Technology: Research on Underrepresentation* (pp. 89–114). https://doi.org/10.7551/mitpress/9780262033459.003.0003

Goode, J., Flapan, J., & Margolis, J. (2018). Computer science for all: A school reform framework for broadening participation in computing. In W. G. Tierney, Z. B. Corwin, & A. Ochsner (Eds.), *Diversifying Digital Learning: Online Literacy and Educational Opportunity* (pp. 45–65). Baltimore, MD: Johns Hopkins University Press.

Goode, J., Margolis, J., & Chapman, G. (2014). Curriculum is not enough. In J. Dougherty & K. Nagel (Chairs), *Proceedings of the 45th ACM Technical Symposium on Computer Science Education - SIGCSE '14* (pp. 493–498). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/2538862.2538948

Grimes, S., & Fields, D. A. (2015). Children's media making, but not sharing: The potential and limitations of child-specific diy media websites. *Media International Australia, 154*, 112–122. https://doi.org/10.1177/1329878X1515400114

Guzdial, M. (2003a). A media computation course for non-majors. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '03*, *35*(3), 104–108. https://doi.org/10.1016/j.psyneuen.2014.09.024

Guzdial, M. (2003b). A media computation course for non-majors. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '03*, *35*(3), 104–108. https://doi.org/10.1016/j.psyneuen.2014.09.024

Guzdial, M. (2010a). Does contextualized computing education help? *ACM Inroads*, *1*(4), 4–6. https://doi.org/10.1145/1869746.1869747

Guzdial, M. (2010b). Does contextualized computing education help? *ACM Inroads*, *1*(4), 4–6. https://doi.org/10.1145/1869746.1869747

Guzman, I. R., Stam, K. R., & Stanton, J. M. (2008). The occupational culture of IS/IT personnel within organizations. *ACM SIGMIS Database, 39*(1), 33–50. https://doi.org/10.1145/1341971.1341976

Guzman, I. R., & Stanton, J. M. (2009). IT occupational culture: The cultural fit and commitment of new information technologists. *Information Technology and People*, *22*(2), 157-187. https://doi.org/10.1108/09593840910962212

Haraway, D. (2000). A cyborg manifesto: Science, technology, and socialist-feminism in the late twentieth century. In D. Bell & B. M. Kennedy (Eds.), *The Cybercultures Reader* (pp. 291–324). London: Routledge.

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments*, *1*(1), 1–32. https://doi.org/10.1080/1049482900010102

Hellens, L. A. von, Nielsen, S. H., & Trauth, E. M. (2001). Breaking and entering the male domain: Women in the IT industry. In M. Serva (Ed.), *Proceedings of the 2001 ACM SIGCPR Conference on Computer Personnel Research* (pp. 116–120). New York, NY: Association for Computing Machinery. http://doi.acm.org/10.1145/371209.371535

Hellens, L. A. von, Pringle, R., Nielsen, S. H., & Greenhill, A. (2000). People, business and IT skills. In J. Prasad & W. Nance (Chairs), *Proceedings of the 2000 ACM SIGCPR Conference on Computer Personnel Research - SIGCPR '00* (pp. 152–157). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/333334.333378

Hewlett, S. (2008). *The Athena factor: Reversing the brain drain in science, engineering, and technology* (Harvard Business Review Report No. 10094). Retrieved from http://rachelappel.com/media/downloads/w_athena_factor.pdf

Hewlett, S. A. (2008). Off-ramps and on-ramps: Keeping talented women on the road to success. *Human Resource Management International Digest*, *16*(2), 26–28. https://doi.org/10.1108/hrmid.2008.04416bae.003

Hug, S., Grunwald, D., Lewis, C., Goldberg, D. S., & Feld, J. A. (2012). Engaging computer science in traditional education. In T. Lapidot & J. Gal-Ezer (Chairs), *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '12* (pp. 351–356). New York, NY: Association for Computing

Machinery. https://doi.org/10.1145/2325296.2325377

Hyysalo, S. (2010). *Health technology development and use: From practice-bound imagination to evolving impacts*. New York, NY: Routledge. https://doi.org/10.4324/9780203849156

Ito, M., Baumer, S., Bittanti, M., Boyd, D., Herr-Stephenson, B., Horst, H. A., … Tripp, L. (2010). *Hanging out, messing around and geeking around: Kids living and learning with new medi*a. https://doi.org/10.1080/1369118X.2010.516760

Jenkins, H., Clinton, K., Purushotma, R., Robison, A. J., & Weigel, M. (2006). *Confronting the challenges of participatory culture: Media education for the 21st century*. Retrieved from www.macfound.org.

Kafai, Y., & Burke, Q. (2014). *Why children need to learn programming*. Retrieved from https://books.google.com/books?hl=en&lr=&id=eskIBAAAQBAJ&oi=fnd&pg=PP1&dq=kafai+and+burke,+2014&ots=p-MCg5xzvF&sig=42h2U8KBAT7I3MfI9wmlBIdXYCI

Kafai, Y., Fields, D. A., & Searle, K. (2014a). Electronic textiles as disruptive designs: Supporting and challenging maker activities in schools. *Harvard Educational Review*, *84*(4), 532–556. https://doi.org/10.17763/haer.84.4.46m7372370214783

Kafai, Y., Fields, D., & Searle, K. (2014b). Electronic textiles as disruptive designs: Supporting and challenging maker activities in schools. *Harvard Educational Review*, *84*(4), 532–556. https://doi.org/10.17763/haer.84.4.46m7372370214783

Kanter, R. M. (1977). Some effects of proportions on group life: Skewed sex ratios and responses to token women. *American Journal of Sociology*, *82*(5), 965–990. https://doi.org/10.1086/226425

Kanter, R. M. (2006). Some effects of proportions on group life: Skewed sex ratios and responses to token women. *Small Groups: Key Readings*, *82*(5), 37–54.

https://doi.org/10.4324/9780203647585

Kaptelinin, V., & Nardi, B. (2012). Activity theory in HCI: Fundamentals and reflections. *Synthesis Lectures on Human-Centered Informatics*, *5*(1), 1–105. https://doi.org/10.2200/S00413ED1V01Y201203HCI013

Karanasios, S., Allen, D., & Finnegan, P. (2015). Information systems journal special issue on: Activity theory in information systems research. *Information Systems Journal*, *25*(3), 309–313. https://doi.org/10.1111/isj.12061

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming. *ACM Computing Surveys*, *37*(2), 83–137. https://doi.org/10.1145/1089733.1089734

Knobel, M., Lankshear, C., Jacobson, E., Jenkins, H., Lewis, M., Luckman, S., … Tufano, N. (2010). *DIY media: Creating, sharing and learning with new technologies*. New York, NY: Peter Lang.

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. *2004 IEEE Symposium on Visual Languages - Human Centric Computing*, 199–206. https://doi.org/10.1109/VLHCC.2004.47

Koehler, M. J., & Mishra, P. (2005). What happens when teachers design educational technology? The development of technological pedagogical content knowledge. *Journal of Educational Computing Research*, *32*(2), 131–152. https://doi.org/10.2190/0EW7-01WB-BKHL-QDYV

Koehler, M. J., Mishra, P., & Yahya, K. (2007). Tracing the development of teacher knowledge in a design seminar: Integrating content, pedagogy and technology. *Computers and Education*, *49*(3), 740–762. https://doi.org/10.1016/j.compedu.2005.11.012

Kumar, D. (2014). Disrupting the cultural capital of brogrammers. *ACM Inroads*, *5*(3), 28–29.

https://doi.org/10.1145/2655759.2655765

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*(4), 429–458. https://doi.org/10.2190/BKML-B1QV-KDN4-8ULH

Kuuti, K. (1996). Activity theory as a potential framework for human-computer interaction research. In B. A. Nardi (Ed.), *Context and Consciousness: Activity Theory and Human-computer Interaction* (pp. 17–44). Retrieved from https://books.google.com/books?hl=en&lr=&id=JeqcgPlS2UAC&oi=fnd&pg=PA17&dq=kuuti+1996&ots=e-egTCu-Dy&sig=UkWqGryLKSuj4p7JKzC4-gmUbFo#v=onepage&q&f=false

Lagesen, V. A. (2007). The strength of numbers: Strategies to include women into computer science. *Social Studies of Science*, *37*(1), 67–92. https://doi.org/10.1177/0306312706063788

Lagesen, V. A. (2008). A cyberfeminist utopia?: Perceptions of gender and computer science among malaysian women computer science students and faculty. *Science Technology and Human Values*, *33*(1), 741–763. https://doi.org/10.1177/0162243907306192

Lagesen, V. A. (2012). Reassembling gender: Actor-network theory (ANT) and the making of the technology in gender. *Social Studies of Science*, *42*(3), 442–448. https://doi.org/10.1177/0306312712437078

Lagesen, V. A. (2016). Gender and professional practices in software engineering. In U. Jørgensen & S. Brodersen (Eds.), *Engineering Professionalism* (pp. 233–253). https://doi.org/10.1007/978-94-6300-752-8_12

Lambert, D. (2018, September 27). *California improves access to computer science courses, but* *still has work to do, according to new research*. Retrieved from https://edsource.org/2018/california-improves-access-to-computer-science-courses-but-still-has-work-to-do-according-to-new-research/602963

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge, UK: Cambridge University Press.

Leonardi, P. M. (2011). When flexible routines meet flexible technologies: Affordance, constraint, and the imbrication of human and material agencies. *MIS Quarterly*, *35*(1), 147–167. https://doi.org/10.2307/23043493

Lin, B., & Serebrenik, A. (2016). Recognizing gender of stack overflow users. In M. Kim (Chair), *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16* (pp. 425–429). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/2901739.2901777

Lin, C.-C., Zhang, M., Beck, B., Olsen, G., Lin, C.-C., Zhang, M., … Olsen, G. (2009). Embedding computer science concepts in K-12 science curricula. *Proceedings of the 40th ACM Technical Symposium on Computer Science Education - SIGCSE '09*, *41*(1), 539–543. https://doi.org/10.1145/1508865.1509050

Linn, M. C. (1992). How can hypermedia tools help teach programming? *Learning and Instruction*, *2*(2), 119–139. https://doi.org/10.1016/0959-4752(92)90027-J

Loewenberg Ball, D., Thames, M. H., & Phelps, G. (2008). Content knowledge for teaching. *Journal of Teacher Education*, *59*(5), 389–407. https://doi.org/10.1177/0022487108324554

Margolis, J., Estrella, R., Goode, J., Holme, J., & Nao, K. (2008). *Stuck in the shallow end: Education, race, and computing*. Cambridge, Mass: MIT Press.

Margolis, J., & Goode, J. (2016). Ten lessons for computer science for all. *ACM Inroads*, *7*(4), 52–56. https://doi.org/10.1145/2988236

Margolis, J., Goode, J., & Chapman, G. (2015). An equity lens for scaling. *ACM Inroads*, *6*(3), 58–66. https://doi.org/10.1145/2794294

Margolis, J., Goode, J., Chapman, G., & Ryoo, J. J. (2014). That classroom "magic." *Communications of the ACM*, *57*(7), 31–33. https://doi.org/10.1145/2618107

Marowka, A. (2018). On parallel software engineering education using python. *Education and Information Technologies*, *23*(1), 357–372. https://doi.org/10.1007/s10639-017-9607-0

Marshall, C., & Rossman, G. B. (2016). *Designing qualitative research* (6th ed.). Thousand Oaks, CA: SAGE PUBLICATIONS.

Martin, F. (2015). Getting my two cents worth in: Access, interaction, participation and social inclusion in online news commenting. *#ISOJ, the Official Research Journal of ISOJ*, *5*(1), 80–105. Retrieved from https://www.researchgate.net/profile/Joshua_Scacco/publication/275644032_Digital_divisions_Organizational_gatekeeping_practices_in_the_context_of_online_news/links/5541a1ed0cf2b790436be454/Digital-divisions-Organizational-gatekeeping-practices-in-the-cont

Martin, P. (2003). "Said and done" versus "saying and doing": Gendering practices, practicing gender at work. *Gender and Society*, *17*(3), 342–366. https://doi.org/10.1177/0891243203017003002

Mayer, R. E. (1976). Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order. *Journal of Educational Psychology*, *68*(2), 143–150. https://doi.org/10.1037/0022-0663.68.2.143

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Communications of the ACM*, *29*(7), 605–610.

https://doi.org/10.1145/6138.6142

McGee, K. (2018). The influence of gender, and race/ethnicity on advancement in information technology (IT). *Information and Organization*, *28*(1), 1–36. https://doi.org/10.1016/j.infoandorg.2017.12.001

McLuhan, M. (2006). The medium is the message. In M. G. Durham & D. Kellner (Eds.), *Media and Cultural Studies: Keyworks* (pp. 107–116). Retrieved from https://books.google.com/books?hl=en&lr=&id=I8dPhB88Sx4C&oi=fnd&pg=PA107&dq=McLuhan+%26+Fiore,+1967&ots=CE-FkE6zgT&sig=kitLhfpxDN23KYTpkD06f5pW4QY#v=onepage&q&f=false

Mellström, U. (1995). *Engineering lives: Technology, time and space in a male-centred world* (Doctoral dissertation). Retrieved from Center for Research Libraries Global Resources Network. (Accession No. 792860925).

Miller, L. A. (1974). Programming by non-programmers. *International Journal of Man-Machine Studies*, *6*(2), 237–260. https://doi.org/10.1016/S0020-7373(74)80004-0

Miner, A. S., Milstein, A., Schueller, S., Hegde, R., Mangurian, C., & Linos, E. (2016). Smartphone-based conversational agents and responses to questions about mental Health, interpersonal violence, and physical health. *JAMA Internal Medicine*, *176*(5), 619–625. https://doi.org/10.1001/JAMAINTERNMED.2016.0400

Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record*, *108*(6), 1017–1054. https://doi.org/10.1111/j.1467-9620.2006.00684.x

National Center for Education Statistics. (2017). *Digest of Education Statistics, 2017*. Retrieved from https://nces.ed.gov/programs/digest/d17/tables/dt17_208.20.asp

Naur, P. (2007). Computing versus human thinking. *Communications of the ACM*, *50*(1), 85–94. https://doi.org/10.1145/1188913.1188922

Ong, M., Wright, C., Espinosa, L., & Orfield, G. (2011). Inside the double bind: A synthesis of empirical research on undergraduate and graduate women of color in science, technology, engineering, and mathematics. *Harvard Educational Review*, *81*(2), 172–209. https://doi.org/10.17763/haer.81.2.t022245n7x4752v2

Papert, S. (1972). Teaching children thinking. *Innovations in Education & Training International*, *9*(5), 245–255. https://doi.org/10.1080/1355800720090503

Papert, S. (1980). *Mindstorms: Children, computers and powerful ideas*. New York, NY: Basic Books.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*(2), 137–168. https://doi.org/10.1016/0732-118X(84)90018-7

Perković, L., Settle, A., Hwang, S., & Jones, J. (2010). A framework for computational thinking across the curriculum. In R. Ayfer & J. Impagliazzo (Chairs), *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '10* (pp. 123–127). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/1822090.1822126

Porter, A., McMaken, J., Hwang, J., & Yang, R. (2011). Common Core Standards. *Educational Researcher*, *40*(3), 103–116. https://doi.org/10.3102/0013189x11405038

Pulimood, S. M., & Wolz, U. (2008). Problem solving in community: A necessary shift in cs pedagogy. *Sigcse '08*, *40*(1), 210–214. https://doi.org/10.1145/1352135.1352209

Rasmussen, B., & Håpnes, T. (1991). Excluding women from the technologies of the future? A case study of the culture of computer science. *Futures*, *23*(10), 1107–1119. https://doi.org/10.1016/0016-3287(91)90075-D

Rennert-May, C., Jurisson, C., Franke, B., Wildeman, B., Spaltro, F., Settle, A., & Hansen, R. (2012). Infusing computational thinking into the middle- and high-school curriculum. In T. Lapidot & J. Gal-Ezer (Chairs), *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education - ITiCSE '12* (pp. 22–27). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/2325296.2325306

Resnick, M., Flanagan, M., Kelleher, C., MacLaurin, M., Ohshima, Y., Perlin, K., & Torres, R. (2009). Growing up programming. In D. Olsen (Chair), *Proceedings of the 27th International Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '09* (pp. 3293–3296). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/1520340.1520472

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Rich, L., Perry, H., & Guzdial, M. (2004a). A CS1 course designed to address interests of women. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education - SIGCSE '04*, *36*(1), 190–194. https://doi.org/10.1145/971300.971370

Rich, L., Perry, H., & Guzdial, M. (2004b). A CS1 course designed to address interests of women. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education - SIGCSE '04*, *36*(1), 190–194. https://doi.org/10.1145/971300.971370

Roldan, M., Soe, L., & Yakura, E. K. (2004). Perceptions of chilly IT organizational contexts and their effect on the retention and promotion of women in IT. In M. Tanniru (Chair), *Proceedings of the 2004 Conference on Computer Personnel Research Careers, Culture, and Ethics in a Networked Environment - SIGMIS CPR '04* (pp. 108–113). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/982372.982399

Rolls, L., & Relf, M. (2006). Bracketing interviews: Addressing methodological challenges in qualitative interviewing in bereavement and palliative care. *Mortality*, *11*(3), 286–305. https://doi.org/10.1080/13576270600774893

Sackett, P. R., DuBois, C. L. Z., & Noe, A. W. (1991). Tokenism in performance evaluation: The effects of work group representation on male-female and white-black differences in performance ratings. *Journal of Applied Psychology*, *76*(2), 263–267. https://doi.org/10.1037/0021-9010.76.2.263

Sammet, J. E. (1972). Programming languages: History and future. *Communications of the ACM*, *15*(7), 601–610. https://doi.org/10.1145/361454.361485

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. *IEE Proceedings - Software*, *149*(1), 24–39. https://doi.org/10.1049/ip-sen:20020202

Schein, E. H. (2010). *Organizational culture and leadership* (4th ed.). San Francisco, CA: Jossey-Bass.

Schilt, K. (2006). Just one of the guys?: How transmen make gender visible at work. *Gender and Society*, *20*(4), 465–490. https://doi.org/10.1177/0891243206288077

Settlage, J. (2013). On acknowledging PCK's shortcomings. *Journal of Science Teacher Education*, *24*(1), 1–12. https://doi.org/10.1007/s10972-012-9332-x

Settle, A., Goldberg, D. S., & Barr, V. (2013). Beyond computer science: Computational thinking across disciplines. In J. Carter (Chair), *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (pp. 311–312). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/2462476.2462511

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher*, *15*(2), 4–14. https://doi.org/10.3102/0013189X015002004

Shulman, L. S. (1987). Knowledge and teaching: Foundations of the new reform. *Harvard Educational Review*, *57*(1), 1–22. https://doi.org/10.17763/haer.57.1.j463w79r56455411

Sime, M. E., Arblaster, A. T., & Green, T. R. G. (1977). Structuring the programmer's task. *Journal of Occupational Psychology*, *50*(3), 205–216. https://doi.org/10.1111/j.2044-8325.1977.tb00376.x

Simpson, J. C. (2001). Segregated by subject. *The Journal of Higher Education*, *72*(1), 63–100. https://doi.org/10.1080/00221546.2001.11778865

Smith, D. C., Cypher, A., & Schmucker, K. (1996). Making programming easier for children. *Interactions*, *3*(5), 58–67. https://doi.org/10.1145/234757.234764

Solomon, C. J., & Papert, S. (1976). A case study of a young child doing turtle graphics in LOGO. In H. Garner (Chair), *Proceedings of the June 7-10, 1976, National Computer Conference and Exposition on - AFIPS '76* (pp. 1049–1056). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/1499799.1499945

Soloway, E., Ehrlich, K., & Bonar, J. (1982). Tapping into tacit programming knowledge. In J. Nichols & M. Schneider (Chairs), *Proceedings of the 1982 Conference on Human Factors in Computing (CHI '82)* (pp. 52–57). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/800049.801754

Spasser, M. A. (1999). Informing information science: The case for activity theory. *Journal of* *the American Society for Information Science*, *50*(12), 1136–1138. https://doi.org/10.1002/(SICI)1097-4571(1999)50:12<1136::AID-ASI17>3.0.CO;2-0

Spender, D. (1997). The position of women in Information Technology - or who got there first and with what consequences? *Current Sociology*, *45*(2), 135–147. https://doi.org/10.1177/001139297045002008

Spinuzzi, C. (2011). Losing by expanding: Corralling the runaway object. *Journal of Business and Technical Communication*, *25*(4), 449–486. https://doi.org/10.1177/1050651911411040

Storey, M. A., Zagalsky, A., Filho, F. F., Singer, L., & German, D. M. (2017). How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, *43*(2), 185–204. https://doi.org/10.1109/TSE.2016.2584053

The National Academies Press. (1999). *Being fluent with information technology*. https://doi.org/10.17226/6482

The National Academies Press. (2002). *Technically speaking: Why all Americans need to know more about technology*. https://doi.org/10.17226/10250

TheFeministSoftwareFoundation. (2014). *C plus equality*. Retrieved https://github.com/TheFeministSoftwareFoundation/C-plus-Equality

Todd, K., Mardis, L., & Wyatt, P. (2005). We've come a long way, baby! In C. Murnan & K. Wainwright (Chairs), *Proceedings of the 33rd Annual ACM SIGUCCS Conference on User Services - SIGUCCS '05* (pp. 380–387). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/1099435.1099521

Trauth, E. M. (2002). Odd girl out: An individual differences perspective on women in the IT profession. *Information Technology & People*, *15*(2), 98–118.

https://doi.org/10.1108/09593840210430552

Trauth, E. M., Quesenberry, J. L., & Morgan, A. J. (2004). Understanding the under representation of women in IT. In M. Tanniru (Chair), *Proceedings of the 2004 Conference on Computer Personnel Research Careers, Culture, and Ethics in a Networked Environment - SIGMIS CPR '04* (pp. 114–119). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/982372.982400

Tsukamoto, H., Takemura, Y., Oomori, Y., Ikeda, I., Nagumo, H., Monden, A., & Matsumoto, K. I. (2016). Textual vs. visual programming languages in programming education for primary school children. In S. Frezza, D. Onipede, K. Vernaza, & M. Ford (Chairs), *Proceedings of the 2016 IEEE Frontiers in Education Conference (FIE)* (pp. 1-7). Erie, PA: IEEE. https://doi.org/10.1109/FIE.2016.7757571

Turco, C. J. (2010). Cultural foundations of tokenism: Evidence from the leveraged buyout industry. *American Sociological Review*, *75*(6), 894–913. https://doi.org/10.1177/0003122410388491

Turner, R. (2013). Programming languages as technical artifacts. *Philosophy and Technology*, *27*(3), 377–397. https://doi.org/10.1007/s13347-012-0098-z

Urban-Lurain, M., & Weinshank, D. J. (2000). Is there a role for programming in non-major computer science courses? *30th Annual Frontiers in Education Conference. Building on A Century of Progress in Engineering Education. Conference Proceedings (IEEE Cat. No.00CH37135)*, *1*, T2B/7-T2B11. https://doi.org/10.1109/FIE.2000.897590

Vanderborght, B. (2018, March). Technology is not neutral [from the editor's desk]. *IEEE Robotics and Automation Magazine*, *25*(1), 4–4. https://doi.org/10.1109/MRA.2017.2787218

Vasilescu, B., Capiluppi, A., & Serebrenik, A. (2014). Gender, representation and online participation: A quantitative study. *Interacting with Computers*, *26*(5), 488–511. https://doi.org/10.1093/iwc/iwt047

Vasilescu, B., Posnett, D., Ray, B., van den Brand, M. G. J., Serebrenik, A., Devanbu, P., & Filkov, V. (2015). Gender and tenure diversity in GitHub teams. In B. Begole & J. Kim (Chairs), *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15* (pp. 3789–3798). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/2702123.2702549

Wacjman, J. (1991). *Feminism confronts technology*. University Park, PA: Pennsylvania State University Press.

Wajcman, J. (2009a). Feminist theories of technology. *Cambridge Journal of Economics*, *34*(1), 143–152. https://doi.org/10.1093/cje/ben057

Wajcman, J. (2009b). The feminization of work in the information age. In D. G. Johnson & J. M. Wetmore (Eds.), *Technology and Society: Building Our Sociotechnical Future* (pp. 459–474). Cambridge, MA: The MIT Press.

Wajcman, J., & Martin, B. (2002). Narratives of identity in modern management: The corrosion of gender difference? *Sociology*, *36*(4), 985–1002. https://doi.org/10.1177/003803850203600410

Webster, J. (2014). *Shaping women's work: Gender, employment and information technology*. https://doi.org/10.1177/030981680107400113

Weinberger, C. J. (2004). Just ask! Why surveyed women did not pursue IT courses or careers. *IEEE Technology and Society Magazine*, *23*(2), 28–35. https://doi.org/10.1109/MTAS.2004.1304399

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

Wentling, R. M., & Thomas, S. (2009). Workplace culture that hinders and assists the career development of women in information technology. *Information Technology, Learning & Performance Journal*, *25*(1), 25–43. https://doi.org/10.1520/GTJ20120061

West, C., & Zimmerman, D. H. (1987). Doing gender. *Gender & Society*, *1*(2), 125–151. https://doi.org/10.1177/0891243287001002002

Wilson, B. C. (2002). A study of factors promoting success in computer science including gender differences. *Computer Science Education*, *12*(1–2), 141–164. https://doi.org/10.1076/csed.12.1.141.8211

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

Wolz, U., & Pulimood, S. M. (2007). An integrated approach to project management through classic cs III and video game development. *ACM SIGCSE Bulletin*, *39*(1), 322–326. https://doi.org/10.1145/1227504.1227422

Wolz, U., Stone, M., Pulimood, S. M., & Pearson, K. (2010). Computational thinking via interactive journalism in middle school. In G. Lewandowski & S. Wolfman (Chairs), *Proceedings of the 41st ACM Technical Symposium on Computer Science Education - SIGCSE '10* (pp. 239–243). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/1734263.1734345

Woodfield, R. (2002). Woman and information systems development: Not just a pretty (inter)face? *Information Technology & People*, *15*(2), 119–138. https://doi.org/10.1108/09593840210430561

Yoder, J. D. (1991). Rethinking tokenism: Looking beyond numbers. *Gender & Society*, *5*(2), 178–192. https://doi.org/10.1177/089124391005002003

Yoder, J. D. (1994). Looking beyond numbers: The effects of gender status, job prestige, and occupational gender-typing on tokenism processes. *Social Psychology Quarterly*, *57*(2), 150–159. https://doi.org/10.2307/2786708

Zimmer, L. (1988). Tokenism and women in the workplace: The limits of gender-neutral theory. *Social Problems*, *35*(1), 64–77. https://doi.org/10.2307/800667

APPENDIX A

Informed Consent Form

PEPPERDINE UNIVERSITY

INFORMED CONSENT FOR PARTICIPATION IN RESEARCH ACTIVITIES

(Re-)Contextualizing Programming to Increase Participation in Computing: K-12 Teachers' Perceptions and Experiences of Integrating Computing into Non-Computational Spaces

You are invited to participate in a research study conducted by Joseph Chipps, M.Ed., and Linda Polin, Ph.D., at Pepperdine University because you are a K-12 teacher who integrates programming in your non-CS classroom. Your participation is voluntary. You should read the information below, and ask questions about anything that you do not understand, before deciding whether to participate. Please take as much time as you need to read the consent form. You may also decide to discuss participation with your family or friends. If you decide to participate, you will be asked to sign this form. You will also be given a copy of this form for you records.

PURPOSE OF THE STUDY

The purpose of the study is to learn more about teachers who integrate computer programming into a more engaging landscape than traditional computer science classes. We are especially interested in how teachers start to use programming and what supports they have on their journey.

STUDY PROCEDURES

If you volunteer to participate in this study, you will be asked to engage in a couple of research procedures. All procedures are voluntary and you can opt out of one or all of them at any point in time.

- **INTERVIEW** We will conduct a face-to-face or remote interview (depending on your location and comfort) for one hour. The interview questions will ask about your experiences as a teacher. The interview will be audio-recorded. The recording will only be heard by members of the research team and will not be played publically. If you do not wish to be audio-recorded, you may choose to not participate in the study with no penalty. After the interview is transcribed, you will be given the choice to look over the transcript and redact or edit information.

- **ARTIFACT DOCUMENTATION** During the interview, we will ask you to share some of your lesson plans or documentation of lessons.

POTENTIAL RISKS AND DISCOMFORTS

The potential and foreseeable risks associated with participation in this study include fatigue or eye strain. You may feel frustrated as you describe your working environment. If at any time you need to suspend or terminate the interview, no questions will be asked, and you will not be penalized.

POTENTIAL BENEFITS TO PARTICIPANTS AND/OR TO SOCIETY

By describing your experience as a teacher, it is anticipated that you may reflect on your practice and benefit from the connections that you otherwise would not have made. There are several anticipated benefits to society which include: providing further information for schools of education that want to support 21st century learning; supporting K-12 teachers who are interested in using programming in their class but lack the tools and means of doing so; and future students who will begin to view computing as a part of 21st century skills rather than an elite subject. Benefits to society are contingent upon the results.

CONFIDENTIALITY

I will keep your records for this study confidential as far as permitted by law. However, if I am required to do so by law, I may be required to disclose information collected about you. Examples of the types of issues that would require me to break confidentiality are if you tell me about instances of child abuse and elder abuse. Pepperdine's University's Human Subjects Protection Program (HSPP) may also access the data collected. The HSPP occasionally reviews and monitors research studies to protect the rights and welfare of research subjects.

The audio recording will be stored on a password protected hard drive in the principal investigator's place of residence, located in a locked safe. The data will be stored for a minimum of three years. Only members of the research team will have access to the audio recordings. During the transcription process, identifiable information will be coded. Pseudonym keys will be kept in the safe separate from the transcripts. After the interview is transcribed, you will be given the choice to look over the transcript and redact or edit information. Transcripts will replace any identifiable information with pseudonyms.

Artifact documentation will be edited in order to code identifying information with pseudonyms.

PARTICIPATION AND WITHDRAWAL

Your participation is voluntary. Your refusal to participate will involve no penalty or loss of benefits to which you are otherwise entitled. You may withdraw your consent at any time and discontinue participation without penalty. You are not waiving any legal claims, rights or remedies because of your participation in this research study.

ALTERNATIVES TO FULL PARTICIPATION

The alternative to participation in the study is not participating.

EMERGENCY CARE AND COMPENSATION FOR INJURY

If you are injured as a direct result of research procedures you will receive medical treatment; however, you or your insurance will be responsible for the cost. Pepperdine University does not provide any monetary compensation for injury

INVESTIGATOR'S CONTACT INFORMATION

I understand that the investigator is willing to answer any inquiries I may have concerning the research herein described. I understand that I may contact Joseph Chipps at [redacted] or Linda Polin at [redacted] if I have any other questions or concerns about this research.

RIGHTS OF RESEARCH PARTICIPANT – IRB CONTACT INFORMATION

If you have questions, concerns or complaints about your rights as a research participant or research in general please contact Dr. Judy Ho, Chairperson of the Graduate & Professional Schools Institutional Review Board at Pepperdine University 6100 Center Drive Suite 500

Los Angeles, CA 90045, [redacted] or [redacted].

SIGNATURE OF RESEARCH PARTICIPANT

I have read the information provided above. I have been given a chance to ask questions. My questions have been answered to my satisfaction and I agree to participate in this study. I have been given a copy of this form.

<u>AUDIO</u>

□ *I agree to be audio-recorded*

□ *I do not want to be audio-recorded*

____

Name of Participant

____

Signature of Participant Date

SIGNATURE OF INVESTIGATOR

____

Signature of Investigator Date

I have explained the research to the participants and answered all of his/her questions. In my judgment the participants are knowingly, willingly and intelligently agreeing to participate in this study. They have the legal capacity to give informed consent to participate in this research study and all of the various components. They also have been informed participation is voluntarily and that they may discontinue their participation in the study at any time, for any reason.

____

Name of Person Obtaining Consent

____

Signature of Person Obtaining Consent Date

APPENDIX B

Interview Protocol

My name is Jake Chipps. I am a doctoral student in the Educational Technology program of Pepperdine University. This interview is being done as part of my dissertation research, and your participation in the interview is completely voluntary. Your real name will not be recorded or used in the study, and all of your responses will be held confidentially. If we come to a question you'd prefer not to answer, just say so and we'll move to the next question. If at any time you wish to end the conversation, you need only say so, and I will end the interview.

The reason I've asked to interview you is to learn more about how teachers implement computer science, specifically programming in their non-computational classes. I hope this research will provide valuable insights into how computer science can be integrated into general education classes.

There aren't any right answers to these questions – I'm interested in your opinion. Feel free to explain something you think I might not understand, but I'll ask follow-up questions if I need a term or a process clarified.

I have mailed to you a document previously which covers the rules and procedures which will guide this interview. Did you receive that document and have a chance to review it?

I would like to record this interview rather than attempt to take notes. It will help me pay better attention to you, and I'll have an accurate record to refer back to later. If you would prefer to answer a question "off the record", I can stop the recording, and then resume it later. During my research, the recordings of this interview will be held securely, and then the research is complete, the recordings will be held for a period of five years, and then will be destroyed.
If you fully understood the statements on that document and are willing to continue with the interview portion, please indicate so by stating "I, <name here> agree to be interviewed and recorded for research."

- Before we talk about programming, can you tell me a bit about yourself as a <subject> teacher.
    - How long have you taught
    - Is this your college/cred'l major area?
    - Do you consider yourself very techie? How so?

- I understand you are someone who makes use of programming, coding, with students as part of your <subject> instruction. Can you tell me how you came to do that?
    - Were you mandated within the dept or elsewhere?
    - Did you team with another teacher?
    - Did you get the idea from a conference or book or paper or PD?
    - What appealed to you about it?

- Can you give me a recent example of a time when students needed to use code for class? Can you show me the assignment or can you show me an example of student work?
    - How did that go? Did they need instruction on coding? Did you support that?
    - How much of your in-class time typically gets spent on this? Is it a one-time thing?
    - Are students assessed on their code work? Why / why not?

- I understand that not everything goes as planned during a lesson. Can you tell me about a time when coding became problematic or difficult in your <subject> class?
    - What did you do to accommodate?

- Can you tell me what prompted you to make the choice to integrate programming in your <subject> class?
    - As best as you can tell, have you seen changes in enrollment?
    - How about students' experiences?
    - Do you have any regrets? Or things you might change?

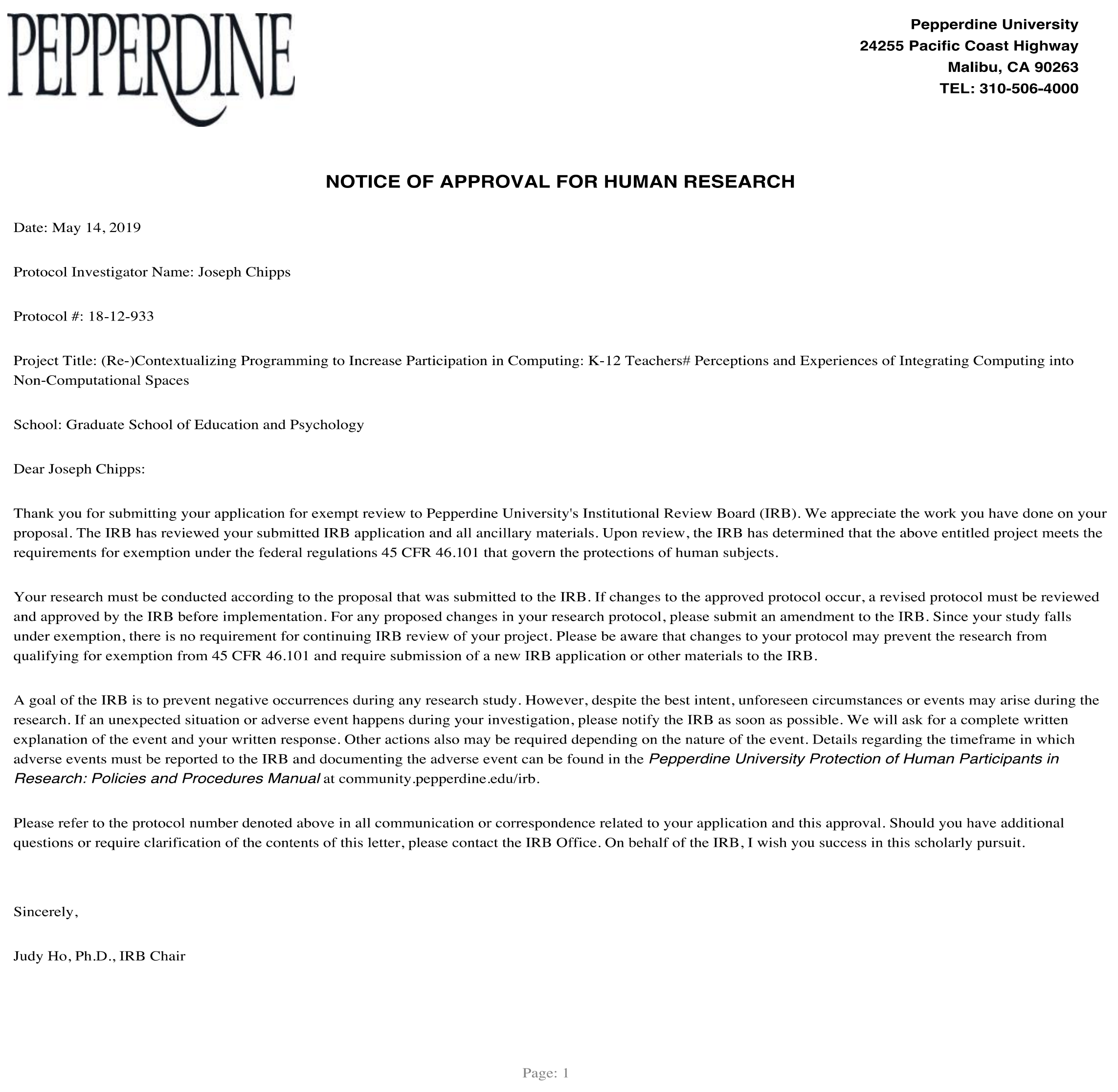
- Can you give me an example of a change you have made in your classroom since you started programming?
    - What prompted the change?
    - Do you do it every day in your class? If not, when do you use it?

- Based on your experience, what would you tell another teacher who wanted to use programming in their classroom?

- I've come to the end of my prepared questions. Considering what we've discussed, is there anything else you would like to say? Have you had experiences that you would like to share that have not been shared yet?

APPENDIX C

IRB Letter of Approval



Pepperdine University
24255 Pacific Coast Highway
Malibu, CA 90263
TEL: 310-506-4000

NOTICE OF APPROVAL FOR HUMAN RESEARCH

Date: May 14, 2019

Protocol Investigator Name: Joseph Chipps

Protocol #: 18-12-933

Project Title: (Re-)Contextualizing Programming to Increase Participation in Computing: K-12 Teachers# Perceptions and Experiences of Integrating Computing into Non-Computational Spaces

School: Graduate School of Education and Psychology

Dear Joseph Chipps:

Thank you for submitting your application for exempt review to Pepperdine University's Institutional Review Board (IRB). We appreciate the work you have done on your proposal. The IRB has reviewed your submitted IRB application and all ancillary materials. Upon review, the IRB has determined that the above entitled project meets the requirements for exemption under the federal regulations 45 CFR 46.101 that govern the protections of human subjects.

Your research must be conducted according to the proposal that was submitted to the IRB. If changes to the approved protocol occur, a revised protocol must be reviewed and approved by the IRB before implementation. For any proposed changes in your research protocol, please submit an amendment to the IRB. Since your study falls under exemption, there is no requirement for continuing IRB review of your project. Please be aware that changes to your protocol may prevent the research from qualifying for exemption from 45 CFR 46.101 and require submission of a new IRB application or other materials to the IRB.

A goal of the IRB is to prevent negative occurrences during any research study. However, despite the best intent, unforeseen circumstances or events may arise during the research. If an unexpected situation or adverse event happens during your investigation, please notify the IRB as soon as possible. We will ask for a complete written explanation of the event and your written response. Other actions also may be required depending on the nature of the event. Details regarding the timeframe in which adverse events must be reported to the IRB and documenting the adverse event can be found in the *Pepperdine University Protection of Human Participants in Research: Policies and Procedures Manual* at community.pepperdine.edu/irb.

Please refer to the protocol number denoted above in all communication or correspondence related to your application and this approval. Should you have additional questions or require clarification of the contents of this letter, please contact the IRB Office. On behalf of the IRB, I wish you success in this scholarly pursuit.

Sincerely,

Judy Ho, Ph.D., IRB Chair

Page: 1