Glass-Boxing Computing With Electronic Textiles: Teaching And Learning With Notional Machines In An Introductory Computing High School Classroom

Gayithri Jayathirtha
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/edissertations

Part of the Teacher Education and Professional Development Commons

Recommended Citation
https://repository.upenn.edu/edissertations/4697

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/edissertations/4697
For more information, please contact repository@pobox.upenn.edu.
Glass-Boxing Computing With Electronic Textiles: Teaching And Learning With Notional Machines In An Introductory Computing High School Classroom

Abstract
Developing a conception of the invisible and abstract internal processes that translate computer programs into observable outcomes is essential yet challenging for learners. Notional machines are simplified notions that educators adopt to make transparent or glass-box program dynamics to learners while teaching. In this thesis, I examined teaching and learning with notional machines during a 14-week online introductory electronic textiles unit in a charter high school. Two broad groups of research questions guided this dissertation—one, exploring teaching, and two, examining student learning with notional machines. Research questions on teaching included: (1) What notional machines did the teacher adopt? (2) What forms did the notional machines take in practice? Research questions on student learning included: (3) How did students interact with notional machines during the unit? (4) Did notional machines support students’ development of computing conceptual agency? If so, how? (5) How did students’ conceptions of computing systems shift after learning with notional machines? Multimodal data—online class recordings, student pre- and post-unit interviews, and student-generated artifacts—were qualitatively analyzed to answer the questions posed. Overall, observational data analysis provided one of the first frameworks to capture notional machines in practice. Notional machines belonged to one of the five themes depending on the electronic textiles concept being simplified and differed along the levels of granularity. Also, notional machines took two distinct representational forms—verbal explanations and participatory roleplays. Analysis of student interactions with notional machines highlighted the agentic roles learners took: questioning, adopting, explaining notions, and roleplaying program execution. Further, student pre- and post-unit interviews indicated that students’ conceptions of program dynamics shifted from being simplistic to more advanced in a set of everyday physical computing devices, showing promise for student sense-making of computing devices outside their immediate programming context. Overall, findings from this study point to future research directions to further explore teaching and learning with notional machines and their potential to expand computing learning beyond classroom contexts.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Education

First Advisor
Yasmin B. Kafai

Keywords
computing pedagogy, electronic textiles, notional machines, physical computing, secondary computing education

Subject Categories
Education | Teacher Education and Professional Development

This dissertation is available at ScholarlyCommons: https://repository.upenn.edu/edissertations/4697
GLASS-BOXING COMPUTING WITH ELECTRONIC TEXTILES: TEACHING AND LEARNING WITH NOTIONAL MACHINES IN AN INTRODUCTORY COMPUTING HIGH SCHOOL CLASSROOM

Gayithri Jayathirtha

A DISSERTATION in

Education

Presented to the Faculties of the University of Pennsylvania in

Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2022

Supervisor of Dissertation

________________________
Yasmin B. Kafai, Professor

Graduate Group Chairperson

________________________
Matthew Hartley, Professor of Education

Dissertation Committee

Mark Guzdial, Professor, University of Michigan

Susan Yoon, Professor

Deborah Fields, Associate Research Professor, Utah State University
Chapter 5. Shifts in high school students' conceptions of sensor-based devices and toys.

Copyright held by International Society of the Learning Sciences.
I would like to dedicate this thesis to all the K-12 computing teachers across the globe. Their efforts to teach abstract computing concepts is the backbone of this work.
ACKNOWLEDGMENT

Though this dissertation and papers published in connection to it may bare only my name, it would not have been possible to do this work if not for the village that supported me throughout this journey. It would have been impossible to do this work if not for the teacher and students participating in the study. I am deeply indebted to my dissertation chair and academic mentor, Dr. Yasmin Kafai, for her constant guidance and encouragement throughout my doctoral program. Learning how to do academic writing was not easy, especially as an international student from India. Many thanks to Dr. Deborah Fields for co-writing and mentoring me throughout my training during the doctoral program. Particular thanks to Dr. Mark Guzdial for thought partnering during my dissertation journey, always nudging me to articulate this thesis better. Last but not least, special thanks to Dr. Susan Yoon for teaching and supporting me throughout my doctoral journey and making me feel comfortable at Penn GSE. I cannot forget the holiday parties at her home and all the food she cooked with particular consideration for each of our dietary restrictions.

Reflecting on my time at Penn GSE, where I learned the what, why, and how of educational research, reminds me of all classes I took, some of which significantly informed this dissertation. Dr. Krystal Strong and Dr. Sharon Ravitch educated me about qualitative methods. Dr. Janine Remillard and Dr. Abby Reisman's classes taught me about teaching and teacher education-related research. And Dr. Alex Posecznick and Dr. Sigal Ben-Porath's classes connected teaching and learning to broader historical, societal, and cultural contexts.

While my professional support's contributions are evident in my thesis, so many invisible hands, minds, and hearts have supported me throughout this journey. Many thanks to my partner, Raghav, who has been there throughout this time, doing everything he can to ensure I successfully complete my dissertation during the pandemic. I was lucky to have my brother, Mohan, by my side, who held the ship together back home in India while I was focused on my thesis thousands of miles away. I cannot thank my parents enough for trusting me since I was a kid and my sister-in-law and parents-in-law who continue to cheer me on. In all of this love and
care I received, I cannot forget my grandparents, especially my grandmother, who taught me to stand by my vision, and each one of my uncles, aunts, and cousins who provided the village I needed to get this far.

Friends here in the US provided the much-needed family away from family. For one, my research team, including Adrian Liu, Mia Shaw, Luis Morales-Navarro, and Justice Walker and Debora Lui in the past, has collaborated during ideation, data collection, and data analysis during my dissertation. I am deeply indebted to Steppingstone Scholars Inc., especially Chris Avery, for funding and helping me pilot online teaching of e-textiles during the summer of 2020. My doctoral cohort at the Teaching, Learning, and Teacher Education division within PennGSE—Shamya Karumbaiah, Mia Shaw, Chenelle Boatswain, Sarah Gudenkauf, Taylor Hausburg, and Kate Miller—who created a strong support community and helped each of us navigate our doctoral trajectories. My friends outside PennGSE at the South Asian Learning Sciences Research Collective, particularly Sugat Dabholkar, Vishesh Kumar, Suraj Uttamchandani, and the Papaya Project, helped me stay afloat and motivated while the world was going through the pandemic and other sociopolitical turmoil. The symposiums and workshops I jointly hosted with friends in these groups helped me do my identity work to find my place in educational research and connect with lovely souls worldwide who continue to inspire me to do educational research.

Despite my best effort to recollect and name people and their relationship to this study, I am sorry if I still miss some. I guess this list can never be complete anytime, especially since everyone connected to me has contributed to this work in some way or another in the last seven years. From casual hallway discussions to intense debates to Zoom catch-ups, several friends and the broader computing research community members have engaged me about my dissertation topic and pushed me to think further and harder. I thank them and acknowledge that this is their work as much as mine.
ABSTRACT

GLASS-BOXING COMPUTING WITH ELECTRONIC TEXTILES: TEACHING AND LEARNING WITH NOTIONAL MACHINES IN AN INTRODUCTORY COMPUTING HIGH SCHOOL CLASSROOM

Gayithri Jayathirtha

Yasmin B. Kafai

Developing a conception of the invisible and abstract internal processes that translate computer programs into observable outcomes is essential yet challenging for learners. Notional machines are simplified notions that educators adopt to make transparent or glass-box program dynamics for learners while teaching. In this thesis, I examined teaching and learning with notional machines during a 14-week online introductory electronic textiles unit in a charter high school. Two broad groups of research questions guided this dissertation—one, exploring teaching, and two, examining student learning with notional machines. Research questions on teaching included: (1) What notional machines did the teacher adopt? (2) What forms did the notional machines take in practice? Research questions on student learning included: (3) How did students interact with notional machines during the unit? (4) Did notional machines support students' development of computing conceptual agency? If so, how? (5) How did students' conceptions of computing systems shift after learning with notional machines? Multimodal data—online class recordings, student pre- and post-unit interviews, and student-generated artifacts—were qualitatively analyzed to answer the questions posed. Overall, observational data analysis provided one of the first frameworks to capture notional machines in practice. Notional machines belonged to one of the five themes depending on the simplified electronic textiles concept and its levels of granularity. Also, notional machines took two distinct representational forms—verbal explanations and participatory roleplays. Analysis of student interactions with notional machines highlighted learners' agentic roles: questioning, adopting, explaining notions, and roleplaying program execution. Further, student pre- and post-unit interviews indicated that students'
conceptions of program dynamics shifted from being simplistic to more advanced in a set of everyday physical computing devices, showing promise for student sense-making of computing devices outside their immediate programming context. Overall, findings from this study point to future research directions to further explore teaching and learning with notional machines and their potential to expand computing learning beyond classroom contexts.
# TABLE OF CONTENTS

COPYRIGHT NOTICE ........................................................................................................... ii

ACKNOWLEDGMENT ........................................................................................................ IV

ABSTRACT ........................................................................................................................ VI

LIST OF TABLES ................................................................................................................ XII

LIST OF ILLUSTRATIONS ................................................................................................ XIII

INTRODUCTION .................................................................................................................. 1

Challenges Learning Computing ....................................................................................... 3

Pedagogical Supports: Notional Machines ...................................................................... 7

Theoretical Frameworks .................................................................................................... 12

Teaching with Notional Machines .................................................................................. 13
  Characteristics of Notional Machines ........................................................................ 15
  Forms of Notional Machines ....................................................................................... 17
  Multimodality of Notional Machines ......................................................................... 18

Learning with Notional Machines .................................................................................. 19
  Studying Interactions through Sociocultural Theories ................................................ 20
  Learning with Notional Machines as a Sociocultural Process .................................... 22
  Learning Conceptions about Computing Systems ....................................................... 24

Research Questions ......................................................................................................... 27

CHAPTER 2 ......................................................................................................................... 31

VIDEO ANALYSIS OF A TEACHER’S USE OF NOTIONAL MACHINES IN AN INTRODUCTORY HIGH SCHOOL ELECTRONIC TEXTILE UNIT: A THREE-TIER FRAMEWORK TO CAPTURE NOTIONAL MACHINES IN PRACTICE ........................................ 31

Chapter Summary ............................................................................................................ 31

Introduction ....................................................................................................................... 32

Background ......................................................................................................................... 34
  Notional Machines ....................................................................................................... 34
  Teaching with Physical Computing ............................................................................. 37
Methodology .................................................................................................................. 39
  Context and Participants .............................................................................................. 39
  Data Collection and Analysis ...................................................................................... 42

Findings .......................................................................................................................... 45
  “The What:” Notions Communicated ........................................................................ 46
  “At What Level:” Level of Granularity .................................................................. 49
  “The How:” Forms of Programs as Plays ................................................................. 52

Limitations and Threats to Validity ............................................................................. 56

Discussion and Conclusion ........................................................................................... 58

CHAPTER 3 .................................................................................................................... 61

A TEACHER’S USE OF BODY, VOICE, AND REPRESENTATIONS TO ENACT
NOTIONAL MACHINES IN CODE EXECUTION ......................................................... 61

Chapter Summary ......................................................................................................... 61

Introduction and Background ....................................................................................... 61

Methodology .................................................................................................................. 63

Findings .......................................................................................................................... 64
  Connecting Program to Circuits: Programming as Controlling Electron Flow .......... 64
  Playing out Program Execution: Program as a Play .............................................. 66

Implications ..................................................................................................................... 68

CHAPTER 4 .................................................................................................................... 69

“HOW DOES THE COMPUTER CARRY OUT DIGITALREAD()?” LEARNER
CONCEPTUAL AGENCY WHILE INTERACTING WITH NOTIONAL MACHINES.... 69

Chapter Summary ......................................................................................................... 69

Introduction .................................................................................................................. 70

Background .................................................................................................................... 73
  Learning with Notional Machines ............................................................................ 73
  Sociocultural Theories of Learning ....................................................................... 75

Methodology .................................................................................................................. 79
  School and Curricular Context .............................................................................. 79
  Participants .................................................................................................................. 84
  Data Collection and Analysis .................................................................................. 85
Discussion ................................................................................................................................. 136
Glass-boxing Computing with Notional Machines ................................................................. 137
Computing Learners as Meaning-making Agents ................................................................. 139
Expanding Computing Learning to Understand Everyday Devices ............................... 141

Conclusions ............................................................................................................................... 142

APPENDIX ................................................................................................................................. 143

BIBLIOGRAPHY ....................................................................................................................... 153
LIST OF TABLES

Table 1: Theoretical Framework to study teaching and learning with notional machines......... 13
Table 2: Structural, functional, and behavioral descriptions of a sensor-based soap dispenser and an interactive toy.................................................................................................................. 26
Table 3: Research questions, data sources, and frameworks guiding this thesis. ..................... 28
Table 4: Spread of focal concepts and pedagogical goals across the four curricular projects within the e-textiles unit.................................................................................................................. 41
Table 5: Notional machine themes, granularities, and forms......................................................... 49
Table 6: Projects, lessons, and notional machines within the revised unit taught by Ben........... 83
Table 7: Notional machine themes, their granularity, and their forms........................................ 83
Table 8: Dates and activities involving student engagement with program dynamics................. 87
Table 9: Analytic and operational categories.............................................................................. 89
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Learners’ interactions with programs and their outcomes.</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Sample e-textile student project consisting of a program and a physical artifact.</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>The role of notional machines in teaching and learning computing.</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Graphical representation of the relationship between notional machines and student conceptions (Fincher et al., 2020).</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Revised graphical representation of the relationship between notional machines and student conceptions, highlighting revisions to the model proposed by Fincher et al. (2020).</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>A sample student project code and physical artifact (above); A screenshot of the online teaching setting on Zoom (below).</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>Visualization of notional machine themes and their distribution throughout the unit.</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>Ben’s shared screen with buggy code (031021); Class arrangement during the roleplay (031621).</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>Notional machine distribution during the four class periods.</td>
<td>53</td>
</tr>
<tr>
<td>10</td>
<td>The multilayered enactment programming as controlling electron flow.</td>
<td>64</td>
</tr>
<tr>
<td>11</td>
<td>The multilayered enactment of program as a play by roleplaying program execution.</td>
<td>68</td>
</tr>
<tr>
<td>12</td>
<td>Fincher and colleagues’ (2020) model for teaching and learning with notional (left); revised model based on sociocultural theories of learning (right).</td>
<td>74</td>
</tr>
<tr>
<td>13</td>
<td>E-textiles construction kit used in the curricular unit (above); Sample student project (below).</td>
<td>81</td>
</tr>
<tr>
<td>14</td>
<td>Representation of categories of student conceptions (left) and a sample e-textiles student project showing code on the computer and a connected physical artifact (right).</td>
<td>114</td>
</tr>
<tr>
<td>15</td>
<td>Pre- and Post-interview artifacts (left) and their generalized representation (center); Snippet of Dave’s response coded for structure (light) and behavior (dark) (right).</td>
<td>118</td>
</tr>
<tr>
<td>16</td>
<td>A visualization of distribution of student responses across structural and behavioral details (table) and themes and frequencies within behavioral explanations pre- and post-unit (graphs either sides).</td>
<td>119</td>
</tr>
<tr>
<td>17</td>
<td>(a) Dave’s annotated drawing of internal structure of an automatic soap dispenser pre-unit; (b) Dave’s pulsometer internals post-unit annotated; (c) Karla’s structural drawing of the bird toy pre-unit (d) Karla’s drawing of fox toy internals post-unit.</td>
<td>121</td>
</tr>
</tbody>
</table>
Figure 18: A representation of the three-tiered framework to capture notional machines in practice.
INTRODUCTION

The introduction of national computing learning programs across the globe—CSforAll in the United States, National Centre for Computing Education in the United Kingdom, and mandatory informatics programs in middle and high schools in Germany, and many other countries—stands as evidence for the recent push towards compulsory computing education within K-12 classrooms (CSTA, 2017; Passey, 2017; Przybylla & Romeike, 2014). In addition to increasing employability, other arguments made across these national policies involve a desire to support learners to better understand and participate in today’s programmed, digital world. Computing education seeks to help learners develop an interest in computing and make connections between their conceptual learning within computing classrooms, and their interaction with technical solutions like smartphones and the internet (CSTA, 2017). With digital devices and technological solutions around us only increasing with time, computing education can provide learners with definite ways of understanding these systems and meaningfully engaging with technologies as informed citizens. Understanding programmed systems by comprehending their internal dynamics becomes especially important to engage with and question the design of tools that perpetuate and amplify social injustices and inequities (Ko et al., 2020). Questioning the design of biased facial recognition software or automatic soap dispensers requires one to understand how data is gathered and processed within these systems and what implications they have on the outcomes of these devices and their users (Benjamin, 2019; Costanza-Chock, 2020). For instance, learning how conditional statements within programs get processed during program execution can provide the knowledge base required to understand how they impact social decision-making when applied in policing or healthcare (Benjamin, 2019). Overall, learning to program computers (i.e., to comprehend, write, and debug programs), is considered as a gateway for learners to learn how a computer works (du Boulay, O’Shea & Monk, 1981) and to critically engage with computing systems around us (Ko et al., 2020).

Increasing introductory computing courses within K-12 classrooms globally calls for special attention to two important aspects: how teachers communicate invisible, abstract
computing ideas to learners, and how learners interact with these ideas and learn about them. More recently, discussions about computing tools that perpetuate and amplify societal injustices further highlight the need for teachers to make computing ideas accessible for learners in ways that develop their agency to question and critique biased designs. Learners within K-12 computing classes need to develop useful conceptions of the inner workings of computing systems and use them while reading, writing, and debugging computer programs. They need to understand data and control flows during computer program execution that cause observed outcomes and apply them to problem-solve. However, the invisible and abstract nature of these processes during computer program execution challenge learners, especially novices.

Notional machines, defined as simplified notions that educators present to make program dynamics accessible to learners through their instructional or tool designs, is one such approach to glass-box computing to learners. Through tool and instruction designs, post-secondary computing teaching and learning is benefiting from notional machine research (e.g., Tunnell-Wilson, Fisler, & Krishnamurthi, 2018). But, we know very little about notional machine usage within K-12 settings, especially where learners get formally introduced to computing for the first time (Fincher et al., 2020). Questions such as, what notional machines K-12 teachers use in practice, what forms they take, how learners interact with them, and how they shift learner conceptions about computing systems, need further investigation.

For this dissertation, I conducted a qualitative study during a 14-week-long high school introductory computing unit taught by an experienced teacher to examine teaching and learning with notional machines, particularly to answer the following research questions: (1) What notional machines did the teacher adopt? (2) What forms did the notional machines take in practice? The second group of research questions on student learning included: (3) How did students interact with notional machines during the unit? (4) Did notional machines support students’ development of computing conceptual agency? If yes, how? (5) How did students’ conceptions of computing systems shift after learning with notional machines? Findings from this research address existing gaps in notional machine research. It provides one of the first accounts of not only the use of
notional machines in a K-12 classroom, but also of student interactions with notional machines within any computing classroom over extended durations. Based on the findings from teacher’s use of notional machines, and students’ interactions and learning from it, this thesis highlights the potential of notional machines in supporting computing learning distributed across time, space, and people.

Below I elaborate on motivations for this thesis and briefly introduce the research questions and ensuing analysis.

Challenges Learning Computing

My research on learning and teaching programming is motivated by the recognition that learners face many challenges learning to program. In addition to understanding the problem at hand, designing solutions and realizing them using a programming language, learners are tasked with learning about the notional system, the syntax and the semantics of the programming language (du Boulay, 1986). The hidden nature of program execution makes it hard for learners to develop a notion of the underlying processes, one that involves interactions across the different layers of abstractions between the source code and the machine (Berry, 2015; Guzdial, 2015, 2017). As shown in Fig. 1, students are limited to engaging with programs (reading, writing, and debugging) and their outcomes (observing and interacting) while the processes that transform program text to outcomes are inaccessible or intentionally black-boxed. The processes underlying the transformation of the program text to observed outcomes on the interface (see Fig. 1) such as making and changing variable values within computer memory during execution is made opaque to learners. Such black-boxing simplifies the dynamic nature of programs while simplifying interactions with programs.

Though black-boxing can help beginner programmers to focus on programs and outcomes without being overloaded with information (Hmelo-Silver & Guzdial, 1996), learners will need to develop simplified yet useful conceptions of internal dynamics to untangle concerns regarding the syntax, semantics, and the conventions associated with particular programming
language and paradigm (Krishnamurthi & Fisler, 2019). Earlier studies have already highlighted novice programmers’ “superbugs”, i.e., associating human-like reasoning abilities to computers while understanding program dynamics, which may not always help them reason programs (Pea, 1986). At the same time, learners are constantly making meaning of program constructs by adopting resources such as mathematical and English language semantics at their disposal to infer their behavior during program execution (Qian & Lehman, 2018). Developing useful conceptions of abstract processes can also help learners develop conceptual agency (Boaler, 2002; Hall & Greeno, 2008) within that programming context, i.e., adopting, questioning, and extending conceptions while reading, writing, and debugging programs (du Boulay et al., 1981), therefore meaningfully engaging with the discipline.

Figure 1: Learners’ interactions with programs and their outcomes.

Recognizing the complexity associated with programming, a wide variety of programming languages and environments have been specifically designed to make computing concepts more accessible and relevant to learners (Guzdial, 2004; Kelleher & Pausch, 2005; also see Interfaces in Fig. 1). Interfaces such as Scratch, for instance, provide a block-based environment for novice programmers, and allow them to make personally-meaningful computational artifacts like animations and games (Resnick et al., 2009). In addition to traditional on-screen programming languages, programming environments have expanded programming to the physical world to include tangible objects such as blocks and robots which provide opportunities for learners to make, program and learn with physical artifacts (Blikstein, 2013; Horn & Bers, 2019). Particularly,
physical computing construction kits require learners to design and create microcontroller-based circuitry before programming them to achieve the desired outcome (O’Sullivan & Igoe, 2004). For instance, electronic textile construction kits (hereafter e-textiles) allow learners to design and craft fabric-based artifacts and sew electronic components such as microcontrollers and lights using conductive threads (Buechley et al., 2013; See to the right of Fig. 2 for an example project). Fabric-based computing artifacts are accompanied by computer programs (as shown in Fig. 2, left) with different statements such as variable definitions and blocks such as setup() and user-defined functions mapping onto the underlying behavior of the circuits. For instance, variable definitions can correspond to the microcontroller pin numbers used in the circuit, setup() module can assign the pin numbers to serve as output devices during program execution, and user-defined functions can send electric signals to the pins to cause different light patterns.

Interacting with and designing the hardware that underlies the software within these systems allows learners to make connections between the programmed system and the programming constructs, glass-boxing the system to an extent by rendering the interaction between programs and the hardware relatively transparent (DesPortes & DiSalvo, 2017). Mappings between the circuit changes are visible, for example in the form of varying electric signals at the microcontroller pins, and the corresponding program text gives away clues for possible relationships between program execution and the observed outcome. At the same time, their similarity with real-world technologies affords the potential for learners to draw connections between their curricular projects and functionally-similar real-world technologies (Przybylla & Romeike, 2014), engaging with disciplinary concepts across space and time.
Despite the recent physical and tangible programming languages and environments, student challenges with understanding program dynamics within these systems persist. The introduction of hardware aspects in addition to on-screen software implies learners need to make connections between programs and outcomes across computer screens and hardware (Booth, Stumpf, Bird, & Jones, 2016; DesPortes & DiSalvo, 2019; Jayathirtha, Fields, & Kafai, 2018). Learners are expected to not only understand program execution but its implications for the behavior of physical circuits. With physical computing systems entering K-12 introductory settings, recent studies have taken a closer look at learners’ challenges within newer learning environments such as physical computing environments. Learners have been noted to have informal, naive conceptions of physical computing devices before engaging with computing ideas within formal classroom settings (Jayathirtha & Kafai, 2021a, Jayathirtha & Kafai, 2021b). The black-boxed program dynamics led learners to think of physical computing devices in simplistic ways without accounting for any role of computation. Similar to undergraduate students within introductory courses (DesPortes & DiSalvo, 2019), Jayathirtha and colleagues (2018) found that high school
students had the most difficulty in debugging physical computing artifacts when bugs required the understanding of the interaction between hardware (i.e., circuits) and software (i.e., program code).

However, learners, even when learning programming with physical robots, have benefited from explicit support to develop conceptions of program behavior (Touretzky, Garder-McCune, & Agarwal, 2017). Middle school students were noted to reason programs and predict their outcomes better when they were provided with “laws of computation” i.e., simplified underlying behaviors of program constructs during execution within that programming context (Touretzky et al., 2017). Furthermore, Aggarwal and colleagues (2018) within the same study found that these compilation rules provided a framework and a vocabulary for students to reason the behavior of robots programmed, overall supporting students’ comprehension of programs. However, studies that have explored supporting learners develop simplified notions of program execution are far fewer, particularly within physical computing contexts (Krishnamurthi & Fisher, 2019). There is a need for studies that explore how novices understand the interaction between programs and circuits during program execution and what support they will need to develop useful conceptions of program dynamics and adopt them in problem solving.

Pedagogical Supports: Notional Machines

Blackboxed underlying program dynamics implies a need to intentionally design learning environments that selectively render useful aspects accessible to learners, i.e., glass-box to better support learners (du Boulay et al., 1981; Mayer, 1981). While glassboxing all aspects may unnecessarily reveal more than what the learners may be ready for, doing so selectively may support learners to reason program execution while reading, writing, and debugging programs. Selective glassboxing of underlying processes can take a few different forms. For instance, a programming environment interface may provide visualizations that represent processes such as making and changing variable values in some simplified ways (e.g., DiSessa, 2005). Or, a teacher may simplify the process through metaphorical narratives such as variables as boxes or
labels for values to be stored (e.g., Fincher et al., 2020). Notional machines, defined as simplified notions that educators present to make program dynamics accessible to learners through their instructions or tool designs, is one such approach to glass-box computing to novice learners.

Historically, notional machines have been defined by several scholars and were incrementally refined to emphasize the importance of communicating the connections between programs and outcomes within a particular programming context. One of the first attempts dates back to the 1980s where du Boulay and colleagues (1981) defined them as “an idealized, conceptual computer whose properties are implied by the constructs in the programming language employed” (p. 265). Here, the computer, its properties, and its relationship to the outcomes of transforming programs were emphasized. However, recently, program execution has been central to how notional machines are defined. For instance, Dickson, Brown, & Becker (2020) conceptualized notional machines as “an abstract model of an execution environment” (p. 159). This is similar to Sorva’s (2013) definition of notional machines as “characterize[ing] computers in its role as executor of programs in a particular language or a set of related languages” (p. 2), highlighting the specificity of notional machines for a programming context. While du Boulay and colleagues (1981) focused on a particular language, Sorva extended the definition to include programming languages that can be described using a particular notional machine, hinting at common features across a set of languages. Further, Krishnamurthi and Fisler (2019) extended these connections to programming paradigms, thinking of notional machines as not just having a potential to explain program execution within a programming language but across languages that share a particular programming paradigm. For instance, notional machines used to teach object-oriented programming in C++ may also be extended to teach Java within the same paradigm. Overall, across definitions, the primary emphasis of notional machines is about mapping program execution to outcomes within a particular programming context.

In addition to programming languages and paradigms, yet another significant aspect of notional machines that has been highlighted across studies is improving accessibility to learners.
As seen in Fig. 2, teachers and tool designers can choose to *glass-box* different aspects of program dynamics depending on the learners’ prior experiences with computing (Fincher et al., 2020; Guzdial, Krishnamurthi, Sorva, & Vahrenhold, 2019; Krishnamurthi & Fisler, 2019). Teachers can choose to blackbox circuit-related details in programming language classes (like Dickson et al., 2020 while teaching Python) while engaging with it actively in case of physical computing classes. Along similar lines, discussing the behavior of programs at the compiler level (see layers within computers within Fig. 3) may be relevant for an advanced computing student in a programming language class while those can be blackboxed for beginner high school programmers. Instructors’ choices on what to *glass-box* and what to *blackbox* also depend on pedagogical objectives based on learner contexts. They use real-world analogies more often in introductory computing courses compared to advanced courses where learners will be better equipped to communicate in computing-related terms (Fincher et al., 2020). Also, undergraduate computing instructors have also hypothesized revising notional machines during a course to adjust for student learning and fluency with programs within that context (Fincher et al., 2020). At the same time, notional machines have the potential to solicit a variety of naïve conceptions that learners hold about program execution (Fincher et al., 2020), and support learners to develop useful conceptions about program execution. Such affordances of notional machines for revealing student conceptions is particularly significant for introductory computing education, where novices start with naïve conceptions about program dynamics (Pea, 1986; Sorva, 2013). As seen in Figure 3, learners can adopt notional machines provided by teachers and curricular materials in addition to their personal observations of program outcomes to develop an understanding of how programs get transformed to outcomes in a particular programming context.
Notional machines are not just relevant to students but also are considered an integral part of computing teachers’ pedagogical content knowledge, since it requires teachers to make program dynamics accessible to learners (Fincher et al., 2020). This is based on Shulman’s (1986) framework to understand teaching where he conceived teachers' knowledge base to teach a discipline as consisting of content knowledge, pedagogical knowledge, and pedagogical content knowledge. Of particular attention here is teachers' pedagogical content knowledge, which needs teachers to not only understand disciplinary content but also to communicate the same to novice learners. Treating notional machines as a part of pedagogical content knowledge means that it is visible in practice, as teachers, in their interactions with students, enact notions to support learners make sense of program dynamics. Situated within the teaching-learning context (Hall & Greeno, 2008), notional machines can be thought of as teachers socializing learners to certain ways of thinking and meaning-making of program dynamics within a programming context. This has led scholars to hypothesize notional machines to be computing education’s signature pedagogy (Fincher et al., 2020; Shulman, 2005).
Despite notional machines conceived as mediating teaching and learning within computing classrooms (Shapiro in Guzdial et al., 2019), there are certain gaps in where and how they are being researched. With respect to the context in which notional machines are studied, there are very few investigations of notional machine usage within the burgeoning K-12 computing settings (Fincher et al., 2020). Most design and research efforts have focused on post-secondary contexts, leaving the field with very little understanding about the use of notional machines in K-12 computing classrooms. Lack of attention to K-12 settings has led to the absence of studies about notional machines within physical computing contexts, which are popular within K-12 introductory settings (Krishnamurthi & Fisler, 2019). With program outcomes observed on a physical, microcontroller-based circuit, developing conceptions of program execution within this context and explaining outcomes will require drawing connections between hardware and software.

In relation to how notional machines have been examined, studies that have explored teaching and learning with notional machines have mostly relied on teacher or instructor interviews to generate notional machine accounts instead of examining notional machines in practice (e.g., Fincher et al., 2020). Similar trends are noted in how student learning has been assessed by analyzing student responses to problem sets or think-aloud interviews instead of observing them in classrooms, interacting with notional machines, while developing their own conceptions of program dynamics (e.g., Cunningham, Blanchard, Ericson, & Guzdial, 2017; Tunnel-Wilson et al., 2018). Though student responses to these tasks provide some understanding of learners' conceptions at a particular time, analysis of observational data, particularly interactions between teachers and students as mediated by notional machines, will allow for a deeper understanding of how students interact with notional machines presented and how they may develop the capacity to meaningfully engage with programs while reading, writing, and debugging them. Further, examining the practice over extended periods of time can also shed light on how notional machines may change as students develop conceptions within a context, and how learners' interactions may change as they develop conceptions and conceptual
agency with them. In summary, while notional machines are promising tools for computing classrooms, there is a need for further investigation to understand their role in K-12 settings, particularly what notional machines teachers adopt and how learners learn with them.

Theoretical Frameworks

To address the above-mentioned gaps within notional machine literature, I situated this dissertation study within an introductory computing high school electronic textiles unit (Kafai et al., 2019) to investigate teaching and learning with notional machines. Specifically, to understand teaching with notional machines, I drew on existing literature on teachers’ use of notional machines (Fincher et al., 2020; Flood, DeLiema, & Abrahmanson, 2018; Solomon, Bae, DiSalvo, & Guzdial, 2020), informed by cognitive and embodied learning theories (see Table 1). To understand student learning during this unit with notional machines, I approached it from two perspectives: sociocultural perspectives of learning to study learners’ interactions during the unit (Greeno, 2012; Greeno & Engerström, 2014; Hall & Greeno, 2008), and cognitive perspectives using structure-behavior-function framework (Bhatta & Goel, 1997) to understand student responses to think-aloud interviews. Examined together, they provided the required lenses, first, to identify the different notions that the teacher presented to the learners throughout the unit; second, to examine student interactions with these notions throughout the unit; and third, to capture overall learning outcomes in terms of their conceptions about programmed systems more broadly. Findings from this thesis should not only contribute to our understanding of notional machines within K-12 settings but also push us to think of them as tools supporting learner conceptual agency development while also expanding understanding beyond the immediate curricular context.
Teaching with Notional Machines

Just as the name suggests, “notional” machines go back to psychology of programming studies in the 1970’s where researchers studied the different kinds of errors human programmers made while interacting with the then high-level programming languages such as FORTRAN and BASIC (du Boulay et al., 1981; Fincher et al., 2020; Youngs, 1974). Conceived as the programmer’s understanding of program dynamics as “mental models” that one would exercise while reading, writing, or debugging programs, notional machines designs involved providing static representations, such as graphical drawings, to communicate the model learners can develop of the underlying processes (Fincher et al., 2020). This is best demonstrated in Fincher and colleagues’ (2020) diagram (see Fig. 4) where the teacher interprets a conceptual model in the discipline to create a simplified “notional machine” that the learners make a “personal version” of. Similar patterns are highlighted in the recent review of notional machine studies where the author noted the dominance of psychological and cognitive theories of mental models and learner misconceptions (Sorva, 2013).

Table 1: Theoretical Framework to study teaching and learning with notional machines.

<table>
<thead>
<tr>
<th>Teaching with Notional Machines</th>
<th>Student learning with Notional Machines</th>
<th>Student conceptions about programmed systems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Student interactions with Notional Machines</td>
<td>Sociocultural theories of learning (Vygotsky, 1987; Greeno &amp; Engerström, 2014; Hall &amp; Greeno, 2008)</td>
</tr>
<tr>
<td>Types and forms of NMs in practice (e.g., Fincher et al., 2020)</td>
<td>Conceptual agency (Boaler, 2002; Hall &amp; Greeno, 2008)</td>
<td>Naïve conceptions (Qian et al., 2016; Pea, 1983)</td>
</tr>
<tr>
<td>Multimodal teacher enactment (Flood et al., 2018; Solomon et al., 2020)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conceiving of notional machines as a kind of conceptual model and student learning as developing a mental model has led to a series of design and research efforts, such as visualization tools or static representations, that teachers use in practice (Sorva, 2013; Sorva, Karavirta, & Malmi, 2013 for a review of visualization tools; Fincher et al., 2020 for a catalog of examples). And yet, they miss an important site of action to examine and better understand notional machines—classrooms. Most of the studies have relied on retroactive interviews of instructors to “capture” notional machines. But, in practice, exploration can shed light on the different complexities associated with teaching and learning within classroom settings. Despite the lack of studies that have examined notional machines in practice, we can hypothesize the nature and the different forms they may take based on existing examples of notional machines. Further, studies that have explored program dynamics teaching, in general, allude to the multimodal nature of teaching, which needs to be explored through intentional analysis of notional machines in practice. Below, we draw from extant literature to hypothesize the different aspects of notional machines we may see in practice to inform research in this thesis on teaching with notional machines and some open questions for investigation.
Characteristics of Notional Machines

Notional machines are considered central to computing pedagogy (Fincher et al., 2020). Supporting learners with simplified notions that glass-box only important and relevant aspects of program dynamics is key to supporting learners’ meaning-making within computing classrooms. Fincher and colleagues (2020) also call this “the education of attention,” i.e., teachers draw learners’ attention to certain aspects of program dynamics that will specifically support learners within a programming and learning context. For instance, the process underlying objects, instances, and references, during Java program execution, usually challenge learners within object-oriented programming classes and can be simplified through adoption of notional machines (e.g., Lewis, 2021). This emphasis on keeping concepts accessible to learners is visible across different definitions of notional machines. Berry (2015), in his dissertation study, drew upon du Boulay’s definition of notional machines and discussed them as “providing an easy way of understanding how a particular language or program is executing” (p. 2). This definition not only draws attention to program execution, but more importantly, the idea that notional machines need to be simple or easily accessible to learners. This is similar to Guzdial and colleagues’ (2019) definition of notional machines as a reference “to a behavior description that is accessible to beginners,” (p. 2) calling out the need to make the program dynamics during program execution accessible to novices. This ties very closely with du Boulay and colleagues’ (1981) original call that notional machines “should be conceptually simple, and second, methods should be provided for the novice to observe certain of its working in action” (p. 265).

With making concepts accessible to learners being the central function of notional machines, scholars have discussed how they can be “locally useful” for learners to make meaning of program execution (Reisbeck in Guzdial, 2017, p. 11). Berry also clarifies that “notional machines do not necessarily accurately reflect the exact properties of the real machine; rather it works on a higher conceptual level by providing a metaphorical layer (or, indeed, several such layers) on top of the real machine that are easier to understand” (p. 2). This is comparable
to how instructors in Fincher and colleagues' (2020) interviews have shared designing not just a single notional machine but a series of notional machines to help learners reason program dynamics throughout the course duration. In addition to making concepts simpler, notional machines must also be “comprehensive enough” for learners to apply across various problem-solving situations within a given programming context (duBoulay et al., 1981; Fincher et al., 2020). And yet, most examples we have are “atomic” in nature, i.e., they simplify a particular concept for learners without an explicit pedagogical orientation to support for extended duration like a course or a curricular unit. Examples include teachers presenting variables as boxes holding values, or to parking slots, but with very little conversation about how they hold up over a course duration (Fincher et al., 2020). In summary, notional machines can be thought of as simplified notions about how programs get transformed to outcomes, but with a conscious effort to keep these notions accessible and useful to learners.

In addition to attending to cognitive aspects of learner contexts, such as prior knowledge and experiences in relation to computing ideas, aspects such as the broader pedagogical context and their role in shaping notional machines needs further investigation. For instance, the role of curricular units and learning activities within them and how they may influence notional machines adopted is yet to be investigated. Roleplays, a form of learning activity usually adopted to communicate abstract concepts in computing classrooms (e.g., Andrianoff & Levine, 2002), are one such example, and yet, they are barely explored under notional machines. As Sorva (2013) points out, learning activities may have the potential to make student reasoning visible, and may potentially invite various forms of interactions with notional machines “rather than having them listen to a lecture or view a visualization” (p. 22). But, given the limited theoretical lenses adopted historically, studies examining the contexts in which notional machines play out are almost absent (Sorva, 2013).
Forms of Notional Machines

Notional machine design efforts have taken a few different approaches to represent simplified notions for learners. As Fincher et al. (2020) have cataloged, notional machine designs consist of various "representations or analogies that put a spotlight on those things that are important to look at" (p. 22). One such example is tool design, where simplified notions get codified in interface designs. For example, in Berry’s (2015) or diSessa’s (2001) design of programming interfaces for learners, the designers represent program execution details through simple graphics such as boxes for variables, objects, and instances. As Berry emphasizes, these designs are all incomplete approximations of reality, just enough for learners to understand program execution within their learning contexts. This is similar to how Guzdial and colleagues (2019) called attention to how notional machines will reveal only select aspects of some layers of abstractions but all not, based on what is relevant for learners, to a point of calling notional machines as sort of “lies” that educators tell, i.e., approximations of reality convenient for learners’ contexts.

While interface designers represent notions on-screen, teaching is more dynamic. Constantly interacting with learners and their contexts allows for a broader variety of forms to bring notional machines to life through explanations, analogies, drawings, learning activities, roleplays, short skits, to name a few. This is evident in the array of examples gathered by Fincher and colleagues (2020). Teachers have noted generating hand-drawn representations, such as arrows drawn to highlight control-flow during program execution or memory stacks drawn to indicate changes in variable values during execution of different functions. Another type of form includes analogies to real-world situations, either demonstrated with tangible objects or pictures. Examples of this type include variables described using labeled clothespins, or visualization of parking lots. A few, rare examples were embodied roleplays and gestures to communicate concepts such as method calls and recursive program execution.
Across these different forms, a key concern being addressed is paying attention to the learner context. This is evident in anthropomorphized computers in LOGO manuals written for younger children and their teachers. However, notional machines in undergraduate and graduate programs are closer to mathematical abstractions involving computing-related language (Fincher et al., 2020). Although these notional machines largely differ in their forms, they all seem to account for the prior knowledge of learners. Further, notional machine designs seem to draw from prior experiences outside computing to make computing ideas accessible to learners. This is evident in how learners’ knowledge and experience with everyday systems, such as parking lots, are recruited to communicate ideas such as variables within undergraduate programming classes. All of these communicate program execution in ways that makes these processes accessible to learners within their contexts.

**Multimodality of Notional Machines**

Across the different forms, there is a notable variety in modalities. Hand-drawn and machine-generated representations, verbal explanations, physical artifacts, and embodied enactments to mention a few. While all of these are drawn from examples captured post-hoc from teacher interviews, more nuances can be expected when studying teaching with notional machines in practice. Theory of embodied cognition (Alibali & Nathan, 2012) particularly highlights that teachers not only communicate their notions verbally or through drawings but also through body. Studies within mathematics classrooms, for instance, have shown that teacher gestures play a significant role in communicating abstract ideas.

A few studies within computing education that have investigated teaching of abstract computing ideas have highlighted the role of teacher embodiment in adding another layer of meaning to communicate computing concepts. For instance, Flood and colleagues’ (2018) study illuminated how teaching team members in an after-school program used hand gestures to animate control flow while discussing program execution. Further, Solomon and colleagues (2020) have demonstrated multimodal enactment within the context of an undergraduate
computing class where the teacher not only employed embodied gestures to communicate recursion but also furthered it with hand-drawn representations. This is similar to Kwah’s (2013) observations. Overall, as seen in Fig. 5 (numbered as 1), this calls for an extension of analyzing teaching with notional machines to consider teachers as full-bodied people and that they would employ a variety of gestures to communicate abstract computing ideas.

Figure 5: Revised graphical representation of the relationship between notional machines and student conceptions, highlighting revisions to the model proposed by Fincher et al. (2020).

Learning with Notional Machines

Just as limited theoretical lenses had narrowed research and design efforts to certain forms of notional machines and understanding the role of teachers, they have also had implications for how student learning has been conceived and what aspects of it have been examined. Conceiving learning as students developing mental models of program dynamics within a programming context has led to researchers capturing snapshots of student conceptions at different times during the course of a unit, or only at the end of a course, and analyzing them for computational models or program tracing patterns (Cunningham et al., 2017; Tunnel-Wilson et al., 2018). A similar trend of studying student learning with notional machines is visible in Sorva’s (2013) review, where the author noted the lack of theoretical frameworks outside of psychological
and cognitive theories. This perspective is largely reflected in Fincher and colleagues’ (2020) model where the learner is considered as an individual who will develop a mental model of the abstract dynamic being presented by the teacher. The process is even conceived as learners mostly absorbing the model presented, as illustrated by the unidirectional arrow in the graphic representation (see Fig. 3). However, none of these notions had any study which examined students in interaction with notional machines i.e., how these personal versions were developed. Below I describe the theoretical expansions needed in order to study student interactions with notional machines in classrooms and at the same time highlight that prior studies can help us better understand individual student conceptions of computing systems. Both together can inform student learning with notional machines.

**Studying Interactions through Sociocultural Theories**

Sociocultural theories of learning provide a suitable framework to study learning as happening in interactions (Vygotsky, 1978). Unlike psychological studies, sociocultural theories expand the unit of analysis beyond the individual to include the activity where the interactions are situated (Greeno & Engerström, 2012; Kafai & Proctor, 2022). Recognizing that learning is social and cultural, and embedded in the context implies attention will be given to interactions learners have with the teacher and the peers as mediated by the concepts being presented that serve as tools for reasoning and problem-solving. These theories posit that mental functioning of humans have social origins, i.e., humans think and reason at a social plane as they participate in an activity. It is within these activities that learners get socialized and encultured into ways of engaging within practices that are valued within the community (Greeno & Engerström, 2012). Learners, who are new to the community and its practices, move from the periphery to the center of the community during their participation within the community (Lave & Wenger, 1991). During this process, they develop ways of thinking, doing, and being within the community, i.e., identity and agency to meaningfully engage within the community (Boaler, 2002). Learners develop identities and
agencies as central participants as they “share and appropriate patterns of interaction and interpretation afforded by the community's practices” (Hall & Greeno, 2008, p. 3).

In addition to the focus on the activity, sociocultural theories of learning highlight the mediating role played by socioculturally shaped tools and signs within these activities (Vygotsky, 1978; Wertsch, 1988). The tools are anything but neutral—they have cultural, social and historical significance within a community whose members have historically adopted them to problem-solve within the discipline. These tools become a part of the representational infrastructure that allows members of the community to develop conceptions and socially engage within the activity (Hall & Greeno, 2008). While experts who are at the center of participation within the community may share formal concepts and tools that are condensed as symbols and equations, learners at the periphery may employ more simplified, functional concepts while problem-solving and reasoning within the discipline (Greeno, 2012). These functional concepts, that emerge in interactions within a community engaging in disciplinary practices, are imbued with local meanings that evolve within the community over time. This perspective lends itself to any analysis of student interactions to understand their conceptual learning as they participate in activities within disciplinary communities as mediated by functional concepts (Hall & Greeno, 2008). Functional concepts serve as conceptual tools for learners to think with them and actively participate in the activity or practice that are meaningful within the community.

One important way to assess learner’s participation within activities and community practices is the gradual development of learner conceptual agency to pursue generality within the discipline (Jurow, 2004; Hall & Greeno, 2008). Learners, while engaging with community practices and tools, should develop the ability to adopt representational infrastructure within the discipline for problem-solving and ask and answer questions about the central understandings within the discipline, both across a variety of contexts. Such interactions allow for learners to develop sophisticated ways of understanding while moving towards the center of participation within the disciplinary community. Overall, treating learning as a sociocultural process allows for
an analysis of interactions that lead to conceptual learning and learner conceptual agency development within a learning community.

Learning with Notional Machines as a Sociocultural Process

Notional machines, similar to functional concepts within communities of learners, are situated within classroom contexts. The teacher, with pedagogical responsibility, will modify that formal concept to something functional for the community of learners. Teaching with notional machines can be conceived as scaffolding a community of learners to participate in practices meaningful within computing by adopting meaningful program dynamic conceptions while reading, writing, and debugging programs. Such a scaffolding will involve intentional glass-boxing of certain concepts while blackboxing others (Hmelo-Silver & Guzdial, 1996). While the teacher may present the notional machine as a way to scaffold learners’ meaning-making of program dynamics, they are deeply influenced by different contextual factors. For instance, the degree of sophistication of notional machines depends on the prior computing learning experiences of learners (Fincher et al., 2020; Guzdial, 2017). Further, notional machines are informed by the programming language constructs and paradigms. Taken together, the influence of contextual factors on notional machines and their ability to shape conversations within computing classrooms justify the expansion of the unit of analysis beyond the individual to include the activity where the interactions are situated. Learners interact with notional machines while engaging with community practices such as comprehending code, writing, and debugging and learning to make meaning of program dynamics in certain consistent ways. As they are getting socialized to make meaning of program dynamics, they also move from the periphery to the center of participation while developing identities and agencies as computing problem-solvers and programmers.

Just as the socioculturally shaped tools and signs within these activities, notional machines serve as tools for learners to engage with while learning how to make sense of program dynamics in ways valued within the community. Learners continuously engage with the
functional concepts presented by the teacher to understand and reason program dynamics in relation to the observed outcomes they produce. As Sorva (2013) mentions, a shift in theoretical framing will then mean students are not just absorbing notions presented by the teacher but are actively engaging and making meaning with them. When examining teaching and learning over a course duration, there is room to explore the potential of notional machines to serve as representational infrastructure within commuting classrooms. Comparable to how formal semantics of a language mediates meaning-making among expert programmers, notional machines may allow for historical and social meaning-making among learners, mediating their actions of making sense of program dynamics within activities and practices such as reading, writing, and debugging programs.

An important aspect of developing conceptions about program dynamics is for learners to use them when reading, writing, and debugging programs contexts outside of that specific activity. This implied a need for learners to adopt the conceptual tools presented to them and navigate problem spaces agentically, either while making their own projects or debugging programs in other contexts. They are expected to not just absorb notional machines as models presented to them but to adopt them across different problem-solving contexts (e.g., Cunningham et al., 2017; Tunnel-Wilson et al., 2018), and ask questions and revise their conceptions while moving towards the center of participation within the disciplinary community. Overall, learning with notional machines can be summarized as represented in Figure 5. Notional machines serve as functional concepts for learners within a class to engage with program dynamics. Learners’ sense-making of program dynamics is mediated by the functional concepts and in sociocultural interactions with other learners (see 2 in Fig. 5) and the teacher over extended times (see 3 in Fig. 5). In the process, learners not just absorb the notional machine provided but develop conceptual agency to engage with program dynamics by explaining and extending them, and even questioning them while problem-solving in other contexts (see 4 in Fig. 5). In addition, teachers are not just delivering static meanings about program dynamics but are dynamically
creating and revising simplified notions based on interactions with learners and curricular objectives (5 in Fig. 5).

**Learning Conceptions about Computing Systems**

While examining classroom interactions allows for understanding the process of conceptual development, studies that have analyzed student responses to think-aloud interviews provide valuable snapshots of student learning. These methods have been particularly helpful to capture naïve conceptions that students have held about computing systems and program dynamics. One of the often-discussed examples within naïve conceptions involve learners conflating semantics from natural spoken English language and mathematics with that of the programming language constructs. For instance, Pea (1986) studied middle and high schoolers’ interpretations of Pascal and BASIC programming constructs and found “superbugs” which are naïve conceptions regarding the functions of certain constructs such as *if* and *while* due to close structural mappings with natural language. While prior understanding of these constructs in natural spoken language negatively affected student meaning-making while programming within the English-speaking American contexts, Qian and Lehman (2018) observed that proficiency with the English language supported young Chinese programmers to better comprehend Pascal programs. Furthermore, they also observed that proficiency in mathematics helped learners interpret similar programming constructs better. However, the brief quantitative correlational analysis conducted by Qian and Lehman (2018) failed to provide further details about student meaning-making of the programming constructs. This only implies that more research is needed to understand these conceptions that learners bring into computing classrooms so that educators and research communities can better support learners in developing their conceptions closer to disciplinary knowledge and practices.

Not only in understanding program constructs, beginner learners have demonstrated naïve conceptions about inner workings of physical computing devices despite tangible and accessible components such as buttons and lights (Cederqvist, 2019; Pancratz & Diethelm,
The blackboxing of different components that constitute the computing system, their interconnections, and the runtime dynamics makes the computing aspects invisible and obscure to novice learners. And yet, the ubiquity of these devices in the form of sensor-based systems around us, such as automatic lights and doors or robotic vacuum cleaners, implies that learners have a variety of user experiences and informal conceptions of these systems. In a recent study, Pancratz and Diethelm (2020) observed German middle and high schoolers surfacing a range of informal conceptions about the inner structural composition of a set of physical computing devices, mostly developed from their user experiences. These not only inform students’ conceptual learning with physical computing construction kits, but they also change as learners develop into more advanced participants within computing courses. Particularly, as learners develop conceptual agency, one needs to explore what it may mean to students’ conceptual shifts especially when interviews usually expect them to adopt their ways of thinking and understanding computing systems to problem solve in a new, unknown context.

Physical computing is a special case of designed artifacts, i.e., systems intentionally designed with specific components to achieve a particular goal. Physical computing systems consist of three key dimensions: structure, behavior, and function (SBF) (Bhatta & Goel, 1997). The physical parts of the system make the structural composition, while its overall purpose in terms of inputs and outputs are the functional aspects; and, the causal underlying logic that ties together the different components to cause outcomes accounts for the behavior (Bhatta & Goel, 1997; Jayathirtha & Kafai, 2021a). Some of the first studies conducted by Goel and colleagues (1996, 1997) adopted SBF to explain designed artifacts such as a simple electric circuit: the different parts of the systems (e.g., power source, lights) and interconnections between them were the structural aspects, descriptions of components and the goal or purpose of the systems in terms of inputs and outputs (e.g., a circuit’s purpose to turn on lights) were the functional aspects, and causal explanations to reason about the outcome (e.g., changes in voltage and current causing the light to turn on) were the behavioral aspects of the systems. Extending this framework to understand PCS such as sensor-based soap dispenser and interactive toys (see
(1) the *structural* aspects include the different components and the physical connections between them; (2) the *functional* aspects consist of the roles of each of these components in relation to the overall goal; and (3) the *behavioral* aspects include the underlying logic causing the overall functionality.

**Table 2: Structural, functional, and behavioral descriptions of a sensor-based soap dispenser and an interactive toy.**

<table>
<thead>
<tr>
<th>Artifacts / Dimensions</th>
<th>Sensor-based Stoplight</th>
<th>Interactive toy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structure</strong></td>
<td>Sensor, wires, soap holder, tubes, motor, and a processing unit; wires connecting motor and sensors in the unit.</td>
<td>Sensor or buttons, lights, speaker, a processing unit, connectors connecting inputs and outputs to the unit.</td>
</tr>
<tr>
<td><strong>Behavior</strong></td>
<td>The run-time dynamics of programs at the processing unit receiving input signals from sensors and applying conditional logic to operate motors.</td>
<td>The run-time dynamics of programs at the processing unit receiving human action signals, applying conditional logic and accordingly signaling lights and speakers to respond.</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Sense a hand or an object at the bottom and run motors to release soap based on the underlying logic.</td>
<td>Sense human press and make different light patterns and play music depending on the type of press (hard/soft) based on the underlying logic.</td>
</tr>
</tbody>
</table>

Some recent studies around novices’ informal conceptions of PCS have touched upon certain aspects of SBF although they shed limited light on students’ understanding of the role of computation or how they relate to these systems through their prior experiences. For instance, Cederqvist (2020) examined middle schoolers’ conceptions of PCS such as car remote key and microcontroller-based projects to uncover their structural and functional understanding. In this study, although novices drew from a variety of prior experiences while explaining these systems, the researcher found that most of their understanding pivoted around the visible components such as buttons and lights. This limited students’ ability to acknowledge the presence of computer programs controlling these systems—thereby precluding further investigation around students’ conceptions of the role of computation in shaping underlying behavior of these systems. More
recently, Pancratz and Diethelm (2020) observed a range of ways in which middle and high school students understood the structural aspects of daily-use PCS such as robotic vacuum cleaners and video game consoles. But, this analysis—limited to uncovering students’ structural understanding—revealed very little about how they understood the functional and behavioral aspects of the systems or experiences that they were drawing from. With PCS making a foray into introductory computing programs across age-groups and national contexts, there is a need for more studies to examine students’ user experiences and informal conceptions of these everyday PCS, specifically studies that explore students’ understanding of computation in shaping their function and behavior.

Research Questions

To address existing gaps in our understanding of teaching and learning with notional machines within K-12 classrooms, the proposed thesis will qualitatively examine the observational and interview data from an introductory computing high school e-textile unit from different theoretical perspectives. The first group on teaching consists of are informed by cognitive and embodied theories of learning (Alibali & Nathan, 2012; Fincher et al., 2020), the second group of research questions on student learning takes advantage of both situated and cognitive theories of learning to analyze student learning in interaction and demonstrates the same in explaining the underlying processes of other physical computing devices. See Table 3 for detailed mapping of the theoretical and analytical frameworks, and the data sources analyzed to answer the questions posed.
Table 3: Research questions, data sources, and frameworks guiding this thesis.

<table>
<thead>
<tr>
<th>Research questions</th>
<th>Teaching with NMs</th>
<th>Student interactions with NMs</th>
<th>Student conceptions about programmed systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>What notional machines did the teacher adopt during the unit?</td>
<td>Types and forms in practice (e.g., Fincher et al., 2020) Multimodal teacher enactment (Flood et al., 2018; Solomon et al., 2020)</td>
<td>How did students interact with notional machines during the unit?</td>
<td>How did high school students’ conceptions of computing systems and programs shift after learning with notional machines?</td>
</tr>
<tr>
<td>What forms did they take in practice?</td>
<td>Seven Classroom videos where teacher introduced key concepts through notional machines</td>
<td>Did notional machines support students’ development of computing conceptual agency? If so, how?</td>
<td></td>
</tr>
</tbody>
</table>

In Chapter 2, I share findings from the first group of questions on teaching with notional machines: (1) What notional machines did the teacher adopt while teaching the introductory ECS e-textile unit? (2) What forms did they take in practice? Video recordings of seven online class periods (80 min. each) spread across 14 weeks will be inductively and deductively analyzed (Derry et al., 2010). The class periods where the teacher introduced the class to different aspects of program dynamics were considered for this analysis, especially since the focus is on notional machines taught by the teacher.
In Chapter 3, I report findings from a particularly focused analysis of four of the class period videos that further highlighted the multimodal nature of teaching with notional machines, one involving embodied gestures, on-screen representations, and voice modulations to highlight key concepts around program dynamics within e-textiles. In sum, findings from this analysis will provide one of the first accounts of notional machines within an introductory physical computing classroom during an entire curricular unit. In addition to offering examples of notional machines from a K-12 introductory classroom teaching, i.e., how the teacher glass-boxed program dynamics while teaching physical computing, they will also provide details such as the forms they took and the granularities of the concepts that were attended to in this class. Further, it added to existing literature on embodied teaching practices another dimension of voice modulation, an aspect barely explored within teaching computing.

In Chapters 4 and 5, I present analysis guided by the second group of questions on student learning which has two parts: one about the process of learning and another the outcome or the product of learning with notional machines. Chapter 4 will focus on the process and answer the questions: (1) How did students interact with notional machines during the unit? (2) Did notional machines support students’ development of computing conceptual agency? If yes, how? Video recordings from eight class periods in addition to student artifacts will be analyzed to capture the different ways in which students will interact with notional machines throughout the unit. The eight class periods where particularly learners were given explicit chances to engage with program dynamics were considered for this analysis. Findings from this analysis has the potential to expand the field’s understanding of how students learn with notional machines, for it will be one of the first observational accounts of students’ interaction with notional machines in practice.

Chapter 5 responds to the final question concerning the end product of learning: How did students’ conceptions of computing systems shift after learning with notional machines? To answer this question, ten pre-unit and post-unit interviews with students will be analyzed for qualitative descriptions (Creswell & Poth, 2016; Bhatta & Goel, 1997) of an e-textile project and
another everyday sensor-based device. This analysis will not only highlight learners’ conceptions of underlying behavior of these devices but also illustrate potential expansion of student conceptions beyond the context of e-textiles to discuss computing within artifacts across time and space.

Finally, in Chapter 6, I will summarize and discuss key findings, address possible limitations, and outline implications for K-12 CS education research and practice.
CHAPTER 2

VIDEO ANALYSIS OF A TEACHER'S USE OF NOTIONAL MACHINES IN AN INTRODUCTORY HIGH SCHOOL ELECTRONIC TEXTILE UNIT: A THREE-TIER FRAMEWORK TO CAPTURE NOTIONAL MACHINES IN PRACTICE

This chapter will be published within ACM Digital Library.


Chapter Summary

Learners’ conceptions of program dynamics shape their reading, writing, and debugging of code. But, the invisibility of underlying program behaviors that transform code to outcomes challenges learners. Teachers adopt notional machines, defined as simplified notions about program dynamics, to support learners within computing classrooms. Researchers have gathered notional machine examples by interviewing post-secondary instructors. But, there is a need to capture notional machines in practice within computing classrooms, particularly introductory high school classes where teachers communicate program dynamics to learners with limited to no prior programming experiences. Through a qualitative video analysis of seven online class periods (80 minutes each) across 14 weeks of an introductory physical computing electronic textiles high school unit, this paper answers: (1) What notional machines did the teacher use in practice? (2) At what levels of granularity did they communicate program dynamics? And, (3) What representational forms did they take? The analysis revealed a three-tier framework to capture notional machines in practice. First, notional machines belonged to one of the five themes depending on the layer of abstraction simplified within electronic textiles. Second, they differed along the levels of granularity—individual atoms, program blocks, relations between blocks, or the entire program. Third, they took two distinct representational forms—verbal explanations and participatory roleplays. Overall, the analysis has two-fold contribution: provides a framework for
future research to capture and study notional machines in practice, and, at the same time, presents one of the first accounts of notional machines adopted within a high school introductory physical computing unit.

Introduction

A primary objective of introductory secondary computing education is to help learners develop a conception of layers of abstractions between software and hardware that make computing systems functional (CSTA, 2012). Developing a viable conception of how programs cause outcomes is quintessential for learners to reason about programs while reading, writing, and debugging them (du Boulay et al., 1981). However, the invisibility of internal processes challenges novice programmers across age groups. From storing data to manipulating and controlling the flow of data that determines conditional outcomes, the obscurity of program execution processes make them inaccessible to novices (Lewis, 2012). Even in contexts such as physical computing environments, designed to support learners with a circuit to program, novices have difficulties understanding how programs cause observed outcomes (e.g., Jayathirtha & Kafai, 2021b). They often attribute computers with human-like “intelligent, interpretive” abilities (Pea, 1993) and have simplistic conceptions of physical computing devices in terms of visible input and output components (Jayathirtha & Kafai, 2021a). Such informal conceptions about the internal dynamics of computing systems more often challenge rather than support novices’ learning programming. There is a need to help learners develop more sophisticated and useful conceptions about how programs generate outcomes (e.g., Fincher et al., 2020; Guzdial & duBoulay, 2019).

One way teachers address this challenge is by using notional machines, which can be defined as simplified notions that educators present to make program dynamics accessible to learners through their instructional or tool designs. Introduced in the 1980s (du Boulay et al., 1981), scholars have expanded the field’s understanding by articulating characteristics of notional machines, capturing examples from instructor interviews, and analyzing their roles in computing
classrooms (e.g., Fincher et al., 2020, Guzdial et al., 2019, Tunnell-Wilson et al., 2018). From metaphorical explanations that provide a layer of meaning to understand program dynamics (e.g., variables as storage boxes, labels for values, etc.) to interactive physical manipulatives, notional machines embody a range of meanings and take various forms (e.g., Fincher et al., 2020, Lewis, 2021). Educators use notional machines so frequently that they have been identified as central to pedagogical content knowledge for computing teachers (Fincher et al., 2020). Despite their significance, there are crucial gaps in research on notional machines: (1) Existing studies rely on instructors’ self-reported notional machines, and almost no studies examine observational data to understand notional machines in practice over extended periods (Fincher et al., 2020); (2) While scholars have tried articulating different characteristics of notional machines (Fincher et al., 2020), frameworks to capture and discuss notional machines in practice are barely available; (3) Most studies are situated within post-secondary contexts, and very few consider K-12 classrooms where programming learners program for almost the first time (Fincher et al., 2020; Guzdial & duBoulay, 2019); and, (4) Lack of attention to programming contexts such as physical computing that is widely adopted to introduce K-12 learners to computing (Kafai et al., 2019; Krishnamurthi & Fisler, 2019). Questions around capturing notional machines and their use over longer periods such as—are they reused, how do they evolve, and what forms they take—have remained speculation within the research community (Fincher et al., 2020).

This study addresses the above listed gaps by examining an experienced teacher’s use of notional machines within a high school introductory classroom during 14-week-long physical computing electronic textiles (e-textiles) online unit. Physical computing systems offer a unique context to study notional machines as the tangible circuit renders transparent the machine being programmed (DesPortes & DiSalvo, 2017). For instance, physical computing construction kits such as e-textiles allow learners to design and program microcontroller-based circuits while making fabric-based computing artifacts (Buechley et al., 2013). However, comprehending the underlying program dynamics between programs and circuits is challenging for novice learners
(DesPortes & DiSalvo, 2019; Jayathirtha & Kafai, 2019). Informed by prior studies and personal observations of student struggles, the participant teacher, Ben (pseudonym), used notional machines while teaching the electronic textiles unit online in Spring 2021. For this paper, I collaboratively and qualitatively analyzed online class videos (Angelillo et al., 2007; Derry et al., 2010) across seven class periods (80 minutes each) to answer the following questions: (1) What notional machines did the teacher use in practice? (2) What levels of granularities did they communicate about program dynamics? And, (3) What representational forms did notional machines take?

Background

Notional Machines

Notional machines are integral aspects of computing teachers’ pedagogical content knowledge, i.e., understanding how to represent or communicate disciplinary ideas in ways appropriate for learners (Shulman, 1987). And yet, confusions prevail around their definition and boundaries (e.g., Dickson et al., 2020; Duran, Sorva, & Seppälä, 2021). Described as pedagogical tools (Fincher et al., 2020), scholars have clarified how notional machines are not semantic roles of program behavior, nor students’ conceptual models, nor the on-screen visualizations (e.g., Dickson et al., 2020; Duran, Sorva, & Seppälä, 2021; Fincher et al., 2020). Originally defined as an “idealized, conceptual computer whose properties are implied by the constructs in the programming language employed” (p. 265, du Boulay et al., 1981), notional machines have been defined across several studies since then. Across definitions, a few key characteristics stand out (e.g., Guzdial, 2015; Fincher et al., 2020). First, notional machines are pedagogical, i.e., learners are the audience while educators or tool designers design and implement them. Second, notional machines need to make program runtime dynamics or processes accessible to learners. These simplifications can communicate meanings across varying granularities with respect to programs and their behavior. For instance, they can simplify the runtime behavior of individual program
statements such as variable declaration or blocks of statements within a conditional block or control flow during the entire program execution (e.g., Fincher et al., 2020; Lewis, 2021), comparable to the levels of granularity discussed in the BLOCK model of student program comprehension (Schulte, 2008). Third, the programming environment, the language, and the paradigm adopted for teaching will shape an educator’s attempt to simplify ideas, while keeping them aligned to agreed-upon concepts within the broader programming communities. For instance, notional machines adopted within an object-oriented C++ class will differ when teaching imperative programming with C++ or other programming languages (Krishnamurthi & Fisler, 2019). Further, teacher reflections of the use of notional machines within a course or a curricular unit have highlighted the use of multiple notional machines to communicate a family of concepts within a teaching-learning context. But, there are barely any accounts beyond individual examples, i.e., notional machines that simplify a particular aspect of a program behavior at a specific instance during the course. Fourth and final, notional machines do not communicate any “ground truth,” and instead are approximations of the actual processes to support “locally useful” sense-making among learners (Reisbeck in Guzdial et al., 2019). duBoulay has articulated them as lies about the underlying processes (duBoulay in Guzdial et al., 2019) to highlight their approximated nature. Overall, notional machines can be characterized as simplifications of program dynamics adopted by teachers to support learners within a programming context.

Drawing from and extending the recent work (Fincher et al., 2020), every notional machine can be conceived to consist of three key aspects: the computing notion or phenomenon it is simplifying, the level of granularity in relation to the program, and the representational form it takes. While the notion being simplified is “what” the notional machine “draws attention to” (Fincher et al., 2020), the level of granularity further clarifies “at what” level the notional machine maps onto the programs being discussed, and the representational forms such as teacher explanations or learning activities that it takes is “how” the notional machines are presented to learners. For instance, computers anthropomorphized as “workers” to teaching programming in LOGO to primary school children and their teachers involved simplifying the control flow of the
entire program and was represented as hand-drawn graphics in textbooks and manuals (Fincher et al., 2020). In other examples, dynamic behavior of function definition blocks and their relationship with function calls have been represented through roleplays (Fincher et al., 2020) or array value assignment statements during program execution through analogical images and explanations (Berry, 2015; Fincher et al., 2020). However, notional machine accounts from practice are far fewer in the literature, particularly those that involve alternative forms such as role plays and skits (Sorva, 2013). The very few accounts available illuminate the complexity of teaching with notional machines by highlighting the role of teachers’ embodied gestures, representations, and language while communicating program execution details (Flood et al., 2018; Solomon et al., 2020).

Post-secondary instructor reflections have surfaced the frequent use of notional machines while teaching programming concepts (Fincher et al., 2020). And yet, examination of notional machines in practice within classrooms and for extended times are missing. Furthermore, studies analyzing the use of notional machines within high school computing classrooms where learners are introduced to computing mostly for the first time are far fewer (with an exception of Tunnell-Wilson et al., 2018). As du Boulay shared with Fincher and colleagues (Fincher et al., 2020), learners need different support during a unit. Instructors have reflected on their use of analogical, "real world" notions while teaching beginner programmers, and their adoption of computing-related examples and language while teaching advanced learners (Fincher et al., 2020). This hints at teachers adjusting notional machines as learners’ knowledge evolves during the unit. But, lack of studies exploring notional machines over weeks and months limits our understanding of how notional machines evolve over time. In addition, variations in the programming environments adopted may further differentiate notional machines within introductory high school classrooms. For instance, introductory computing classes within K-12 settings adopt block-based programming and physical computing environments to bypass syntactic demands and invite diverse learners (e.g., Kafai et al., 2019). Associated notional machines may defer from the current accounts from post-secondary contexts that adopt on-
screen programming environments like Java and support learners with prior programming experiences. For example, physical computing systems consist of circuits in addition to programs and notional machines need to support learners’ understanding of interactions between circuits and programs during runtime (more in the next subsection). Gathering such notional machine accounts in practice will clarify their role in computing pedagogy and help build the burgeoning K-12 computing teaching community (Fincher et al., 2020; Shulman, 1986, 2005).

Teaching with Physical Computing

Physical computing construction kits have historically played a central role within introductory K-12 computing education. From Papert’s robotic turtle (1980) to the more recent microcontroller-based tools, introducing young learners to computing has involved a physical artifact to think with, in addition to computer programs on the screen (Blikstein, 2013). Several physical computing construction kits provide learners with hardware tools, modularized to different degrees, to realize program outcomes (DesPortes & DiSalvo, 2017). Kits, such as electronic textiles (e-textiles), provide learners opportunities to craft and program fabric-based electronic artifacts with microcontrollers, output components (e.g., lights, speakers, etc.), and input components (e.g., analog sensors, digital switches, etc.) (Buechley et al., 2013; see Fig. 6, above for a sample student project). Curricular materials recruiting physical computing construction kits usually take constructionist approaches to teach computing by supporting learners to realize functional artifacts and, in the process, learn from experiences (e.g., Kafai et al., 2019).

Despite design efforts to make computing ideas accessible through tangible artifacts within physical computing, novices continue to be challenged with program comprehension or the understanding of the relationship between programs and outcomes. This struggle is evident in DesPortes and DiSalvo’s (2019) study, where novice undergraduate students struggled comprehending a simple program of blinking two LEDs connected to an Arduino board. Not only did the researchers observe that students had naïve conceptions regarding the sequential nature of program execution, but they also noticed students’ struggles with the semantics of specific in-
built function calls, highlighting the need for notional machines to better support novices. Similar observations were noted in Booth and Stumpf's (2013) study, where students, even with some experience with programming, had challenges grasping the semantics of constructs like if-statements within a physical computing programming environment, especially when bugs were distributed between circuits and programs (Booth et al., 2016; DesPortes & DiSalvo, 2019). In sum, there is a need to support novices to understand the interaction between programs and circuits to make programming meaningful for them.

Pedagogical objectives within physical computing involve communicating about programs and their interactions with the circuitry during runtime to produce observed outcomes. Outcomes in physical computing are usually blinking lights or actuators that respond to inputs such as buttons, switches, and sensors on microcontroller-based circuits (see Fig. 6, above). Teaching with physical computing systems requires teachers to go beyond the visible components and support learners to understand internal behavioral details that can help reason interactions during program execution, such as the data and control flow (Ananthanarayan & Boll, 2020). For example, e-textiles hide the internal processes such as variable storage and changes, challenging students while understanding their overall behavior (DesPortes & DiSalvo, 2019; Jayathirtha et al., 2018). However, teachers have to exercise their pedagogical content knowledge to communicate: circuits and their behaviors; programs and their interactions with the circuits; and translation of program texts to observable outcomes on physical artifacts (Przybylla & Romeike, 2014; Sentance, Waite, Yeomans, & MacLeod, 2017). But, we have no accounts of any notional machines that may be adopted while teaching physical computing (Krishnamurthi & Fisler, 2019).
Figure 6: A sample student project code and physical artifact (above); A screenshot of the online teaching setting on Zoom (below).

Methodology

Context and Participants

The study was conducted at a public charter high school located in a U.S. city during Spring 2021 when this school was fully online due to the COVID-19 pandemic. Ben (pseudonym), a teacher with 10+ years of experience teaching high school computing, taught the constructionist physical computing e-textiles unit within the year-long introductory computing course, Exploring Computer
Science (ECS), for the sixth time (Kafai et al., 2019; see Fig. 6, below for a screenshot from the class). Ben was chosen for this exploratory study about notional machines within high school classrooms due to his disciplinary background in computing, extensive experience teaching ECS, and leading teacher professional development sessions for over 5 years. Though he had taught the curriculum in-person for six years before, this was his first time teaching it online and to particularly focus on teaching program dynamics (see Table 4).

During Fall 2020, Ben collaborated with the research team to redesign the curriculum for online implementation based on his experiences teaching online unprepared during Spring 2020. During this time, Ben was also chosen for this exploratory study about the use of notional machines in a high school classroom due to his extensive experience and fluency with teaching the ECS e-textiles unit. Ben and I met six times (~1 hour each time) to discuss student challenges learning with e-textiles based on our joint experiences and the potential use of notional machines to mitigate them. We noted notional machines that Ben had already used in previous years teaching the e-textiles unit (e.g., programs as plays and roleplaying program execution while introducing new programming constructs). We also identified parts of the curriculum where students could benefit from simplified notions of the underlying mechanics (e.g., circuit behavior, relationship between circuits and programs during runtime). However, we did not design specific notional machines during these meetings and instead oriented ourselves towards the goal of supporting learners with simplified conceptions of underlying program dynamics within e-textiles.

During Spring 2021, he taught the curricular unit online where students were formally learning to program for the first time. The class met on the Zoom online meeting platform on alternate days (~80 min. each day) for 14 weeks. The class had 34 students, most of them in their ninth grade (14 years olds); 25 of them and their parents consented to participate in the study (9 identified as female, 16 as male; ethnicity-related data was not collected). I took the role of participant-observer (Erickson, 1985) and served as a teaching assistant (TA), helping with logistics associated with teaching physical computing online during the pandemic (Jayathirtha, Fields, Kafai, & Chipps, 2020), while collecting data for the study.
The constructionist e-textile curricular unit was designed for students to make four fabric-based projects while exploring circuitry and computing ideas (Kafai et al., 2019) (see Fig. 6 for a student project). Across the four projects, the unit progressed from programming simple light patterns to reading digital and analog inputs and causing differential light patterns using conditional logic (see Table 4 for conceptual progression). Regarding the e-textile programming environment, Ben used the Adafruit Circuit Playground microcontroller and sewable electronic components such as lights and conductive thread (see Fig. 6). The class programmed the microcontroller using the text-based Arduino programming language and followed an imperative programming paradigm. Programs in the unit mostly consisted of three parts: global variable declaration and/or definitions, setup() function to assign microcontroller pins specific roles during program execution, and loop() function to program the expected outcome on the circuit, comparable to the main() function in other programming languages except that this repeatedly loops during execution (see project code in Fig. 6). The class used a simple online on-screen programming environment that offered only a code editor and a serial monitor to view data streams whenever needed. Program outcomes such as light patterns were visible on accompanying physical circuits.

Table 4: Spread of focal concepts and pedagogical goals across the four curricular projects within the e-textiles unit.

<table>
<thead>
<tr>
<th>Projects (Days*)</th>
<th>Paper circuit (1-5)</th>
<th>Wristband (6-11)</th>
<th>Mural (12-22)</th>
<th>Human Interaction (31-37)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Foci</td>
<td>Simple circuit</td>
<td>Parallel circuit</td>
<td>Programming with digital inputs</td>
<td>Programming with analog sensors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prgmng. w/ lights</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pedagogical goals</td>
<td>Program execution affects electron flow within circuits and cause outcomes</td>
<td>Circuit connections determine data flow during program execution and shape outcomes</td>
<td>Programs consist of variables, logical expressions, statements, and functions. Circuit components’ function and behavior are determined by program execution</td>
<td></td>
</tr>
</tbody>
</table>

* Instructional days 23-30 were not part of any project, the class discussed debugging strategies.
**Data Collection and Analysis**

Classroom videos captured throughout the unit was the main source of data to capture notional machines employed during the unit. I screen-recorded all the 37 class periods (80 min. each) and wrote end-of-day summaries throughout the 14-week unit. Screen recording captured all the visible aspects of Zoom sessions, including chat window, learner and teacher windows, and any screen shares.

I collaboratively conducted systematic qualitative video analysis of selected seven class periods to identify notional machines adopted and to analyze their characteristics (Angelillo et al., 2007; Derry et al., 2010, Erickson, 2006). Below, I explain the data analysis process stepwise for readability, although the overall process was not as linear. As a step towards recoverability of methodology and resultant findings (Checkland & Holwell, 1998), I share the five-step process with explicit theoretical mappings and examples whenever helpful. Moving from whole-to-parts (Erickson, 2006), I start from all class videos and end with a set of exemplary episodes selected to highlight notional machines and their features.

**Selecting class periods for analysis**

I read all the class period summaries iteratively and identified seven periods when Ben guided learners to think about program dynamics for at least an hour (7 out of 37; see Appendix A). On other days, class time was mostly dedicated to small group or whole class student work with circuit design or making the e-textile projects, as per the curriculum (Kafai et al., 2019). The seven class periods (9, 10, 11, 12, 19, 20, 32) were spread between the last three projects which involved programming.

**Generating content logs**

For each of these seven selected days (dates written in MMDDYY format), I watched the videos repeatedly to generate content logs (Erickson, 2006) with student and teacher utterances (all anonymized), gestures, chat entries, and shared screens and/or documents related to teaching
program dynamics. Producing thick descriptions of the classroom activities was a way to maintain data validity for researcher collaborators who were unfamiliar with the context (Creswell & Miller, 2000). The content logs chunked period-long videos into “events” of definite classroom activity (such as explaining parts of a program, setting up a roleplay, etc.); each event varied in length and lasted between 2 to 20 minutes.

**Deductive identification and analysis of notional machines**

Each event was deductively analyzed to answer the first question posed (Angelillo et al., 2007; Derry et al., 2010). I collaborated with another computing education researcher, external to this project but familiar with e-textiles, to bring alternative perspectives to the analysis (Erickson, 2006). We met weekly over a period of four months to jointly read activity descriptions and analyze them deductively to identify and examine notional machines. Guided by prior literature, our analysis to capture all the nuances associated with notional machines involved constantly: (1) checking for the aspect of program dynamics Ben was teaching, (2) identifying the simplified notion presented in the place of the underlying dynamics, (3) recognizing the mapping between the dynamic nature and the program text, and (4) analyzing to understand the representation form of communication. This led to identifying a total of 75 notional machine episodes across seven days.

**Inductive analysis of notional machines**

We iteratively analyzed each notional machine inductively (Creswell & Poth, 2016). Based on my participant-observer perspective (Erickson, 1985) and the iterative extended engagement with the data, I generated five key themes inductively by repeatedly reading the thick descriptions. I, in addition to the external researcher, jointly coded a subset of notional machines across two of seven days (~35% of total) to belong to one of the five themes. Upon discussion, we generated a codebook with theme definitions and added examples. We then coded the remaining episodes as belonging to one of the five key themes using this codebook (90% agreement), while constantly looking for exceptions. For instance, Ben introduced programs as texts written in “computer
“speak” language for computers to “understand,” and compared variables to how humans “remember” information. Such notional machines were coded as belonging to the “anthropomorphized computer” theme. Some episodes were coded to communicate multiple notions at once. Like, roleplaying “program execution as plays” also “anthropomorphized microcontrollers” by performing human-like reasoning while reading programs and controlling circuit elements.

To further connect notional machines within themes and draw out details of implementation, we inductively analyzed each notional machine episode for their levels of granularity and forms. We jointly examined episodes to categorize the concept being simplified as belonging to one of the levels of granularity w.r.t. the program text (Schulte, 2008). And, we inductively identified two distinct representational forms that Ben employed: verbal explanations and participatory roleplays. Verbal explanations included Ben describing invisible program dynamics through metaphors or analogies to real-world examples; participatory roleplays where Ben invited his students to take on different roles and enact program execution.

Since the levels of granularity and representational forms were visibly obvious, we did not quantitatively calculate reliability (McDonald, Schoenebeck, & Forte, 2019). Instead, we took more collaborative approaches to establish reliability, as done in qualitative research such as video analysis (Creswell & Miller, 2000; McDonald et al., 2019). In order to establish validity of these emergent categories, we shared our themes, forms, and examples during Fall 2022, first with Ben himself for ecological validity and then with another outsider a senior computing education researcher, to bring in different perspectives on the data and the interpretations (Creswell & Miller, 2000); both confirmed our analysis after a few rounds of discussions.

**Selecting exemplary accounts**

While a broad presentation of five themes with example notional machines will answer the first two questions, thick descriptions of specific episodes had to be chosen to present nuances of the
different representational forms, explanations and roleplays, in practice. Among the five themes of notional machines that emerged during this analysis, “program as plays” theme had the broadest distribution in terms of the representational forms the notional machines took and was chosen to answer the third research question posed. Besides, this theme was how Ben supported learners to understand one of the most challenging aspects of physical computing, i.e., interactions between programs and circuits, and it involved forms such as roleplaying that the field has very few accounts of (Fincher et al., 2020; Sorva, 2013).

Findings

Ben employed a mix of notional machines throughout the unit just as previous research had predicted (Fincher et al., 2020). He reused and gradually extended his notional machines as the pedagogical goals in the unit shifted (visible along each row of Fig. 7). This is unlike earlier accounts of notional machines of individual concepts used at a particular time while teaching, as recollected retrospectively by instructors during interviews. Furthermore, Ben’s notional machines to teach physical computing had five main themes spread across different layers of abstractions within e-textiles, communicating programs’ interaction with the visible computer compilation, with the microcontroller-based circuit, and circuit behaviors independent of programs (see left-most part of Fig. 7 for how the themes are distributed across the layers of abstractions). Throughout the unit, notional machines also differed in the different parts of programs, individual statements to blocks to entire programs, that they were simplifying for learners (more in Table 5). With respect to representational forms, Ben wavered between verbally explaining and orchestrating elaborate roleplays to highlight aspects of program execution. Below I will briefly describe different notional machine themes that Ben adopted and their granularities and provide detailed accounts of “Program as Plays” notional machines to illuminate the representational forms.
Ben’s notional machines to teach programming within e-textiles spread across five main themes (see note and left column in Fig. 7): (1) *Anthropomorphizing computers* helped Ben to present simplifications of program-specific interactions with the computer, specially during compile time, (2) *Program as plays* helped him discuss the interaction between programs, computer, and different circuit elements during compile and runtime, (3) *Anthropomorphizing microcontrollers* to communicate interactions between microcontrollers and other circuit elements during program execution, (4) *Microcontroller pins as programmable gates* to connect circuit behavior at the electron level to program execution, and (5) *Circuits as electron loops* to discuss and reason circuit design and behaviors independent of programs, mainly the basic electronic aspects required to understand outcomes at circuits.

Ben presented computers as human-like agents, particularly while describing interactions between programs and computers as seen on the screen. For instance, computers were attributed with human-like characteristics such as to reason and differentiate syntactic and semantic errors and meanings. When introducing programming, Ben introduced the programming language as having "very, very specific [meanings] that the computers understand" (031021). He used this notion to establish specific meanings and language for symbols such as "=", "==", "&&", {}, to separate out any informal meanings students had from meanings that are aligned with how these symbols are processed during compile time. In order to support first-time learners to make sense of the text-based Arduino programs, he discussed programs written in “computer speak”
language (031021), just like how humans communicate and understand particular languages like English. Not only at the beginning of the unit (see Fig. 7 for the spread throughout the unit), Ben continued to present program statements such as digitalRead() in an anthropomorphized sense by comparing the statement execution to humans reading information such as books and gestures (052021).

Ben adopted Program as play set of notional machines while engaging students to understand the mapping between program execution and circuit elements. Though not non-overlapping with the earlier theme (visible in Fig. 7), Ben specifically employed aspects of plays to provide a way of making sense of program execution by comparing program text as play scripts, variables as characters in a play with definite roles, and program execution as the actual drama with characters playing out particular scripts (e.g., 031221, 031621, 041821). The context of programming with microcontroller-based physical circuits in addition to the structure of Arduino programs consisting of three distinct parts rendered themselves to Program as play notional machines. The initial global variable declarations were compared to character introductions at a play, the setup() to assigning them with specific roles (e.g., INPUT, OUTPUT), and the loop() to the actual play unfolding where characters enact as per the script. This theme, starting off with simple programs to reason blinking lights, extended all the way in the unit when students had to program with multiple digital units.

Anthropomorphizing microcontrollers was the way Ben further engaged the class to discuss microcontroller-based circuit behavior in relation to programs. Unlike anthropomorphizing computers that helped communicate interactions between computers and programs, anthropomorphizing microcontrollers allowed Ben to present simplified notions of the relationship between microcontrollers and the different components such as lights, buttons, and sensors during program execution. For instance, this notion was used when discussing how microcontrollers would “send” and “receive” signals while interacting with output and input components (031021), or how microcontrollers would execute conditional statements by
reasoning with logical expressions just as humans (041621). For example, an elaborate explanation compared the execution of conditional statements with “&&” logical operation to children following parents’ instructions to “clean the room and throw the trash” (041621).

To further draw attention to the relationship between the microcontroller pins and programs during execution, Ben introduced and discussed *Microcontroller pins as programmable gates*. Connecting to students’ prior experience of making simple circuits in the first project in the unit (until 031021), Ben discussed microcontroller pins as gates that allow or not electron flow based on program statements such as digitalWrite() during execution. Such approximated explanations were particularly useful to make accessible complex relationships between the two spaces, on-screen programs on computers and the outcome observed on the physical circuit throughout the unit (see Fig. 7). From reasoning programs that cause blinking lights (031021) to understanding how statements like digitalRead() produce streams of numerical data (052021), microcontroller pins as gates allowed Ben to present a simplified notion of how text-based programs cause circuit outcomes during execution.

Finally, *Circuits as electron loops* was the one theme that Ben used to delve into underlying electric behaviors of physical circuit elements independent of programs. This mostly helped the class reason circuit design such as lights having a common ground, lights connected to same microcontroller pins behaving the same way, etc. It also helped the class reason outcomes in terms of program elements, for instance, how a pair of lights connected to the same microcontroller pin would blink together. Current flow within circuits that cause these outcomes was compared to different real-life situations such as the lazy river ride in amusement parks (031221). However, this theme was short-lived (see Fig. 7) as it continued into the microcontroller pins as programmable gates once circuits were connected to microcontrollers and programmed after the first project in the unit.
While capturing differences in notional machines by thinking of them as distributed across the levels of abstractions is one way to understand these pedagogical tools, exploring their levels of granularity is another way of studying the notion being communicated in terms of mapping it onto actual program texts.

“At What Level:” Level of Granularity

Comparable to Schulte’s (2008) BLOCK model proposed for student program comprehension, Ben’s notional machines also could be categorized as communicating program dynamics at a particular level of granularity in relation to program text. This involved presenting simplified ideas about the runtime behavior of atomic aspects of a program such as individual literal or a sentence, program blocks such as the different sections of Arduino programs, relationship between different blocks, and the entire program. Every project involved a loop of introducing relevant atomic elements and moving up to blocks and the entire program before students made their own projects and programmed them. For instance, Mural project involved notional machines introducing digitalWrite(); statement followed by understanding if-conditional blocks and the execution of logical expressions inside them, and discussing sample programs. Below are examples of notional machines across each level of granularity that further elaborates the simplified notion (also see Table 5).

<table>
<thead>
<tr>
<th>Granularity → Themes ↓</th>
<th>Individual statements</th>
<th>Blocks</th>
<th>Relations</th>
<th>Entire Program</th>
<th>Circuit only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthropomorphized Computer</td>
<td>int led = 10; as “led gets the value of 10” as “understood” by the computer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programs as Plays</td>
<td>Variables as character names</td>
<td>setup() as characterizing</td>
<td>Different parts of the Arduino</td>
<td>Execution as play</td>
<td></td>
</tr>
<tr>
<td>Function definition and calls</td>
<td>programs</td>
<td>if-blocks</td>
<td>loop()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
<td>----------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Anthropomorphized microcontroller</strong></td>
<td>digitalRead(); delay(); as reading and causing delays</td>
<td>loop()</td>
<td>if-blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Microcontroller pins as programmable gates</strong></td>
<td>pinMode(); digitalWrite();</td>
<td></td>
<td>Simple blinking light program as constant manipulation of gates</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Circuit as Electron Loops</strong></td>
<td>How statements like digitalWrite(); cause electron-level changes</td>
<td></td>
<td>How do lights turn on? What does power mean?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ben adopted a variety of notional machines to simplify atomic aspects of programs such as individual literals (e.g., HIGH/LOW, INPUT/OUTPUT in case of Arduino) and statements (e.g., digitalWrite() or delay()). For instance, variable definition statements such as “int led = 10;” were discussed as instructions to computers in “computer speak” language, anthropomorphizing computers as making sense of the statement as “led gets the value of 10.” In some cases, multiple notional machines were employed at different times during the unit to emphasize behavioral details at different levels of abstraction. One such example is notional machines to communicate digitalWrite(); statement behavior. Earlier in the unit, when the class was just getting introduced to programs after working with circuits, digitalWrite(); were discussed in terms of the electron loops they cause, leading to the lights to turn on and off (031021). Later during the unit, the same statement was discussed in terms of how they control the microcontroller pins to act as a gate (e.g., 031221, 031821).
Discussing the behavior of certain blocks of programs, once again, happened through various notional machines. For instance, syntactic or logical groups of statements (e.g., loop() and setup() blocks) were initially described as anthropomorphized microcontrollers that are “sending” or “receiving” signals (031021). However, the same blocks were discussed as a part of a play when the class was preparing for roleplay (e.g., 031821). loop() block was simplified to act as a play script that repeatedly gets executed by the actors. Similarly, relationships between program blocks (e.g., conditional statements; relationship between user-defined functions and their calls within the loop() block) also took different meanings during different times in the unit, depending on the level of detail that was suitable for the conversations. Overall, learners were provided with multiple ways of making sense of program dynamics for the same program construct.

Discussions about relationships between blocks of programs soon transformed into engaging with entire programs, just before students had to program their own projects within the unit. For example, discussion about the relationship between loop() and setup() blocks was followed by approximating the runtime dynamics of the entire program in terms of programs controlling the microcontroller pin that would lead to outcomes like blinking lights (031021). With the introduction of new electronic and program elements such as digital inputs during the mural project, Ben facilitated whole class roleplays to highlight the underlying processes connecting programs and circuit components to produce interactive physical artifacts (e.g., 031821, 041621).

While considering the notional machine in terms of levels of abstraction and granularity highlighted the very notion being simplified, i.e., addressing the “what” of the notional machines, analysis of the representational form revealed the ways in which these notions were expressed or “how” the notions were presented to learners as verbal explanations and participatory roleplays. By elaborating on one of the themes that frequently emerged throughout the unit and in different forms, I describe the different nuances associated with the representational forms notional machines under Programs as plays took.
“The How:” Forms of Programs as Plays

*Programs as plays* was a particular theme of notional machines that broadly took both the representational forms, verbal explanations and participatory roleplays, several times throughout the unit (row 2 in Fig. 7). While Ben introduced variable declarations as naming characters in a play, he iterated similar explanations to compare different parts of Arduino programs to their corresponding play elements (e.g., pinMode() as characterizing play characters). Further along the unit, Ben included students in roleplays to discuss program execution dynamics, bringing the play to life while slowing down and pausing for explanations and clarifications occasionally. For readability, below I present select episodes of teaching with notional machines in the same order as Ben taught. Each episode has a summary followed by description of the notional machines that highlights the form they took during enactment.

**Explaining variable definition**

Summary: Ben played out a short skit to explain variable definition in terms of plays when the class met with a buggy scenario while naming a variable on Day 9 (031021). He compared variable name tokens in programs to character names in plays to communicate the importance of consistent variable names for successful program compilation. During this episode of online teaching, Ben modulated his voice and actively used the screen cursor to connect his explanation with different parts of the program.

Description: Ben had changed the variable name from "led" to "white_led" in the declaration statement but not in the rest of the program (Fig. 8). Compiling based on a student’s suggestion, the compiler indicated errors at lines where the variable was called "led." Some students interpreted the error message, “use of undeclared identifier ‘led’,” as the computer not accepting the underscore in the “white_led.” Four students mentioned that “led” should be renamed as “white_led” throughout the program, but Ben wanted to ensure that all students had a way to reason the error and fix it.
Ben, employing a notional machine, provided a simplified way to understand the error. He said “For example, I am going to tell you a story about Sheila [an imaginary character],” and simultaneously highlighted the “white_led.” He then highlighted the “led” in the next line and said, “Now, Carol was a really cool person.” Pointing at students’ confusion, he said, “you thought I was telling you about Sheila (highlighting ‘white_led’). I set you all up to talk about Sheila, but then I was telling you about Carol (highlighting ‘led’).” He compared the variables in a program to characters in a play, and program itself as telling a story in a play. This allowed him to further relate students’ confusion to the computer’s error during the compilation process: “if you were talking about Carol, you need to tell me that Carol’s going to come up. I can’t handle this! So that’s kind of like what a computer does.” Further, to specifically respond to one of the student’s earlier concerns, he clarified that underscores could be used as long as the names are consistent.

**Explaining setup() block**

Summary: Building on the introduction, in the next class (031221), Ben took the opportunity of writing a simple program to explain different parts of the program in terms of plays. He compared them to a play script—global variables as declaring characters in the play, setup() section as setting roles for the characters, and the loop() section as where the play sequence is written—to provide learners a simple way of understanding an Arduino program and its relation to outcomes.
Description: At the beginning of the class, Ben had the task of writing a program to cause a light to blink. Starting with an empty editor on the shared screen, he asked “can someone tell me what character I should create?” Students started typing in “int led = 10;” in the Zoom chat window. He explained the next part of the program, setup() section, as the module “that happens only once where characterization of characters happens” i.e., where the variables can be set up to be either inputs or outputs during program execution. After writing out the pinMode() function statement inside the setup() section, he discussed the loop() section as the part of the program where the “actual thing happens.” He typed out a comment inside the loop() section and invited students to provide him with statements that would make the light blink. He repeatedly brought in these explanations for the few classes (e.g., 031221, 031821). For students who were learning text-based programming for the first time, Ben distinctly explained the three parts of the program and their roles during execution in terms of a play through this notional machine.

Participatory roleplaying of program execution

Summary: Ben involved students and facilitated roleplays while extending the notional machines to further concretize the interactions between microcontroller, circuit components, and programs during runtime (031621, 031821). Unlike verbal explanations where students were expected to listen and occasionally respond to Ben’s questions, roleplaying program execution increased student participation while also making their conceptions about runtime dynamics visible, allowing Ben to pause for clarifications. Repeated throughout the unit a few times, roleplaying program execution became a way of the whole class understanding program execution when new constructs were introduced (e.g., digital inputs in the Mural project).

Description: Ben introduced roleplay as a way to “test out” the program before running it on the actual circuit. The class had written a program to make a light pattern and included two lights named “wristband1” and “wrisband2” connected to two different microcontroller pins (031621). Setting up the roleplay, Ben once again explained the program in terms of characters and their characterization. In addition, he set up for an embodied roleplaying of program execution, starting
with assigning two student groups to act as “wristband1” and “wristband2,” the TA as the microcontroller and Ben himself as “wristband1.” He explained each of their roles in terms of plays. He said that the microcontroller would “execute the code and will be the ‘conductor’ or the director of the play and the students will be the characters in the play.” He asked the lights to follow the microcontroller’s instructions just as “when the director tells you something, you better do it.” He established specific actions for lights to act out their states—closed fists as off and open fists as on.

During the roleplay, all the actors got a chance to enact their understanding of program dynamics during execution. For instance, in the first set of iterations of the roleplay, the TA acting as the microcontroller read program statements sequentially while highlighting line-by-line. Ben and students acted as lights and changed their states based on the binary values sent to them during program execution (see Fig. 8). Connections between code and the light patterns were thus played out as students acting as “wristband1” and “wristband2” opened and closed their fists based on the parameters of digitalWrite() statements executed. The sequential execution was repeated a few times, “going back to the first line” at the end of the loop() section each time.

In the following class (210318), when the program and the circuit included a digital switch input, Ben extended the “play” to include the digital switch as a “new character” but with a different role. While “declar[ing] characters,” Ben introduced the on-board switch as a new character in the play and named it “switchy,” and “characterized” it as an input since “the computer will be receiving information, on or off, from it.” Similar to previous roleplay, Ben repeated roles and responsibilities. For instance, Ben clarified that the lights cannot change their status or stop until the microcontroller tells them to. Further, he iterated that Dave, a volunteer student acting as a variable storing switch state, was “going to read the only when instructed,” i.e., when the microcontroller is processing the digitalWrite() statement.
During this iteration, Ben acted as the microcontroller but occasionally stepped out of it to provide clarifications and explanations. For instance, Dave, despite Ben’s earlier instructions, changed the value he was storing as soon as the switch toggled even when the microcontroller was not executing the digitalWrite() statement. Ben immediately paused the roleplay to remind Dave that he could not change since he was not asking him to read and update the value. Further, when Ben was executing conditional statements, students acting as lights halted once Ben moved the control outside the block. Ben paused again to tell “don’t stop waving” while he continued to read. When iterating over the loop() section for the second time, Ben reminded the lights to “keep waving, we are not done yet.” This time, Dave steadily held to the value he read even when the TA flipped the switch value. After a couple more iterations without any pauses for explanations, Ben called it an end and said “this will go on forever,” once again iterating the looping action of the block.

Roleplaying notional machines allowed Ben to explain program execution details unlike explanations in a few different ways: he was able to include learners in the process of communicating the underlying program dynamics, observe student actions for their conceptions about program execution, and adjust his explanations and orchestration accordingly. Further Programs as plays theme allowed him to explain dynamic behavior across different levels of granularity, from individual statements to the entire programs. In sum, participatory roleplays afforded a distinct set of affordances by making student thinking more visible compared to explanations. But, Ben coupled them together and weaved them with the unit throughout to support learners alongside the evolving concepts.

Limitations and Threats to Validity

The findings reported come with limitations that future studies can address and further contribute to our understanding of notional machines. First, this study was conducted in an online setting due to the ongoing COVID-19 pandemic. The class met on Zoom, which offered very limited ways
of capturing teaching practices and student interactions with notional machines. While we know teaching in physical classrooms affords several opportunities to observe teacher gestures and how they communicate program-related ideas (Flood et al., 2018; Solomon et al., 2020), online teaching limits all the observations to a box on the screen. Even among students, only those who turned on their video, audio, or engaged in the chat were visible during the class. With the ongoing pandemic, the teacher and the school desisted from mandating any kind of participation (e.g., compulsory video on). This challenged data collection of students who did not or could not participate actively on Zoom. Unlike physical classrooms with videos that captured teacher and student presence anyway, data collection within online settings resulted in capturing very few student interactions, therefore making the observations about notional machines’ evolution during the unit somewhat limited. Similar observational study located within in-person settings can address this limitation and illuminate the role of student engagement in shaping notional machine life cycles within curricular units.

Another limitation was that the data was collected in a single high school introductory class. Although the data over 14 weeks allowed for an in-depth qualitative analysis of notional machines in practice, this was also a particular teacher’s notional machines, shaped by his experiences and disciplinary background, at the very least. Future research can explore notional machines across classrooms and teachers and further explore the connections between aspects such as teacher expertise that may shape teaching with notional machines. Research examining notional machines adopted by teachers with different experience levels can further help us understand the relationship with teachers’ notional machines and their pedagogical content knowledge (Shulman, 1986).

Lastly, this exploratory study was also one of the first attempts to study notional machines by analyzing observational classroom video data. Lack of prior efforts and established frameworks meant the research team spent significant time in drawing out the qualitative nuances and connecting the complexities of classroom teaching with notional machines while inductively
generating the framework. This also included constant member-checking and collaborative approaches to understanding the phenomenon. Considering this as the first step, future research can adopt the proposed framework and move towards employing multiple coders and establishing statistical measures to infer validity.

Discussion and Conclusion

Revisiting the questions posed, Ben used multiple notional machines in different forms spread across five key themes during the unit. They were aimed to support students make sense of underlying processes during program runtime across the level of circuits and programs. Further, they differed in granularity in relation to the programs the notional machines were simplifying: it ranged from providing meaning to a single token to a statement to an entire program. And, representations took two main forms, verbal explanations and participatory roleplays, affording distinct opportunities for learners to interact with program dynamics. The above account not only confirms scholars' prediction that teachers use a set of notional machines during a unit but also shows how these notions evolve over time and adopt different forms (Fincher et al., 2020). Overall, the three-tiered approach helped capture all the notional machines, their characteristics, and their evolution during a curricular unit.

This was one of the first attempts to capture notional machines in practice and develop a theoretical framework to capture the different characteristics of these pedagogical tools. The three-tier approach to capturing notional machines has highlighted some of the key characteristics such as the very notion being simplified as a feature distinct from the form it takes. This clarifies some of the ongoing confusions in understanding what notional machines are and how we can capture them in practice (Dickson et al., 2020; Duran et al., 2021). Having such a framework also enabled to surface the dynamic nature of notional machines. Unlike the traditional way of thinking about notional machines as static computing models, this framework allowed to capture its dynamic nature, one that evolves in interaction between teachers and students.
Adding the dimension of granularity to Fincher and colleagues' framework (2020) further separated out the ideas communicated. Future research can use this framework and see what other revisions may be needed.

This was one of the first accounts of notional machines in practice, particularly within a physical computing unit (Krishnamurthi & Fisler, 2019). Teaching computing within physical computing environments involves not only programs but their interactions with circuits. As seen in the accounts above, notional machines spread across different layers of abstractions within this programming environment. However, these notional machines were also embedded within e-textiles that offer a particular modular aspect to physical computing. The visible uninsulated conductive thread that connects the microcontroller with actuators required the teacher to communicate electron-level behaviors for students to reason the circuit outcomes. However, future research on capturing notional machines should look at more modular toolkits such as robotic environments (DesPortes & DiSalvo, 2017) and their relationship with notional machines used.

Introductory computing courses such as ECS are introducing young learners to computing for almost the first time. Studying notional machines within introductory contexts like these, just like this study, will provide notional machine accounts different from the current examples in the community, predominantly emerging from post-secondary contexts (Fincher et al., 2020). Undergraduate and graduate students usually have some prior experience with programming and the course objectives differ from introductory high school classes. These introductory settings can be seen as stepping stones for learners to understand the field of computing, ways of thinking, and being in the field. Pedagogy within these settings shape novices' thinking about computing in profound ways. As Shulman (2005) points out, it is in these settings that signature pedagogy communicates "what counts as knowledge in the field and how things become known" (p. 54). Notional machines, as discussed above, precisely do that—they communicate to novices what it means to understand computing within physical computing and
how they can discuss and engage with the field while creating artifacts. The pervasiveness of notional machines within computing classrooms (Fincher et al., 2020) in addition to interactive student participation clarifies the key role notional machines play in shaping students’ conceptions about underlying program dynamics.

Providing learners with models and language to observe, reason, and engage with program transformation within computing systems is crucial to supporting learners within computing classrooms (duBoulay et al., 1981). Overall, notional machines involved providing students with ways of understanding the invisible, underlying processes that caused observed outcomes in computing systems.
CHAPTER 3

A TEACHER’S USE OF BODY, VOICE, AND REPRESENTATIONS TO ENACT NOTIONAL MACHINES IN CODE EXECUTION

Chapter Summary

Making disciplinary ideas accessible to learners is central to design efforts within learning sciences. Notional machines are simplified notions about internal processing of programs that educators present to learners through the instructional, tool, or interface designs in ways appropriate for their learners. In this chapter, I report from qualitative video analysis of four online class periods [a total of 320 minutes] where an experienced high school teacher enacted a set of notional machines within a 14-week-long introductory computing electronic textiles unit. Elaborating on two episodes, I will illuminate the multimodal teacher enactment involving gestures, voice, and on-screen representations, and discuss implications for examining teaching practices.

Introduction and Background

Design efforts within learning sciences have historically focused on making disciplinary ideas accessible to novices. Several examples stand out across disciplines—teaching complex system ideas through metaphorical roleplay (e.g., Danish, 2014) or teaching electromagnetism by employing augmented reality (e.g., Yoon, Anderson, Lin, & Elinich, 2017). Within computing education, learners need similar support to develop a conception of invisible processes translating static code to observable outcomes in order to read, remix, write, and debug programs (Fincher et al., 2020). Novices’ struggles understanding the internal dynamics, for instance, interactions between circuits and programs (e.g., Jayathirtha & Kafai, 2021b), call for pedagogical tools to support learning.

Notional machines are pedagogical tools to make computer program execution details accessible to novices (Fincher et al., 2020). They involve expanding control and data flow
processes that transform static code text into observed outcomes in computational artifacts. “Drawing attention to or making salient the hidden aspects of computing” (p. 22) is the primary role of notional machines (Fincher et al., 2020). These designs focus on particular computing concepts, omit extraneous information, and include helpful layers to make ideas accessible to novices. Notional machines take various forms—machine-generated representations within programming environments and curricular materials, or teacher-generated metaphorical explanations such as variables as parking spaces, to name a few (see examples cataloged by Fincher et al., 2020).

Prior explorations of teacher enactment of notional machines within in-person teaching-learning settings have alluded to its multimodal nature. For instance, informed by embodied cognition literature, earlier work has studied the role of teacher’s gestures, gaze, object manipulation in bringing to life processes such as control flow during program execution (Flood et al., 2018; Kwah, 2013). In addition to the embodiment, Solomon and colleagues (2020) have also shown how hand-drawn representations such as box drawings add another layer of meaning to communicate concepts. This chapter extends the literature on teacher enactment of disciplinary ideas by highlighting additional layers of voice modulation and on-screen representations within an online physical computing unit. The recent shift to online teaching led to the adoption and manipulation of representations on shared screens within computing classes (Jayathirtha et al., 2020), adding another mode of engagement within classrooms.

I analyzed an experienced high school teacher enacting a series of notional machines during 14-week-long online electronic textiles (e-textiles) unit within an introductory computing course (Kafai et al., 2019). I paid close attention to the multimodal teacher enactment across the four selected class periods (Derry et al., 2010) to understand how the teacher made computing concepts accessible to novices. In addition to focusing on teacher gestures (e.g., Flood et al., 2018), I attended to voice modulations and on-screen representations as the teacher enacted notional machines.
Methodology

The study was conducted at a public charter high school located on the U.S. west coast. An experienced high school computing teacher, Ben, taught online a constructionist physical computing e-textiles unit within a year-long introductory computing course, *Exploring Computer Science*, for the sixth time (Kafai et al., 2019). Learners made four fabric-based tangible projects and gradually explored circuitry and computing concepts. Learning to program in the context of e-textiles required learners to design microcontroller-based circuits, sew them onto fabric artifacts, and program them with text-based Arduino language to realize desired functionalities such as light patterns that respond to button presses or sensor inputs (see Focal content row in Fig. 9).

I iteratively read summaries of all the 37 class periods (80 min. each) to identify days when Ben introduced most concepts during the unit. Video analysis of 4 out of 37 class periods (9th, 12th, 20th, 32nd) involved repeatedly watching screen recordings of class periods in four passes: (a) cleaning the auto-generated transcripts; (b) generating video logs with timestamps and notes on teacher gestures; (c) enriching notes with emergent aspects of enactment: voice modulation and on-screen representations; (d) jointly analyzing video logs with annotated transcripts with another researcher to identify notional machine episodes. To capture voice modulations, the other researcher and I developed a symbol system to indicate different variations within the transcript (* when emphasizing, _ for slowing down, ~ for switching voices to change perspectives).

Altogether, we identified five notional machine strands across 26 episodes during the four class periods (dark blocks in Fig. 9; dates in MMDDYY format). Ben adopted metaphors such as circuits as flowing electron loops and programs as controlling them to discuss the circuit behavior in relation to program execution. Further, he associated computers and microcontrollers with human-like characteristics such as reading, remembering, and writing to demonstrate control and data flow during program execution (Fig. 9, left column). Below we will describe two episodes that best highlight the multilayered and multimodal nature of teacher enactment.
Figure 9: Notional machine distribution during the four class periods.

<table>
<thead>
<tr>
<th>Date (Lesson)</th>
<th>031021 (Hacking Wristband)</th>
<th>031821 (Programming w/ switch)</th>
<th>041621 (Programming w/ 2 buttons)</th>
<th>052021 (Programming w/ sensors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Content / Notional Machines</td>
<td>• Introduction to programming e-textile artifacts</td>
<td>• Connections between circuit &amp; microcontroller</td>
<td>• Digital logical expressions &amp; conditional statements</td>
<td>• Comparisons &amp; conditional execution</td>
</tr>
<tr>
<td></td>
<td>• Variable declaration and definitions</td>
<td>• Designing a simple microcontroller-based circuit</td>
<td>• Program execution w/ 2 buttons</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Program execution to generate light patterns</td>
<td>• Program execution w/ a switch and a light</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circuits as flowing electron loops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anthropomorphizing Computer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anthropomorphizing Microprocessor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming as controlling electron flow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program as a Play</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Stamps</td>
<td>00:10 00:30 00:50 01:10</td>
<td>00:10 00:30 00:50 01:10</td>
<td>00:10 00:30 00:50 01:10</td>
<td>00:10 00:30 00:50 01:10</td>
</tr>
</tbody>
</table>

Findings

Overall, notional machine strands were interweaved during enactment (Fig. 9). Below we will elaborate episodes at two time-frames to illuminate the multimodal enactment involving gestures, voice modulation, and on-screen representations (shaded blocks on 031021 and 031821 in Fig. 9).

**Connecting Program to Circuits: Programming as Controlling Electron Flow**

For the first time, Ben introduced to the class how programs translate to light patterns on e-textile artifacts (shaded block on 031021 in Fig. 9; see Fig. 10 to follow the episode). By now, learners had connected their e-textile wristband to the microcontroller. Ben had provided a starter code that would make the wristband lights connected to pin#10 on the microcontroller blink upon running the program (code on the shared screen on Fig. 10).

Explaining the overall goal of the task, Ben modulated his voice as he said: "So, the *pins* are going to be your power. And, we are going to tell *pin 10* what we ~want it to do~" (53:50, 031021 video). He stressed two key parts of the task: the pin and its number. Further, he changed his voice to sound mechanistic to highlight that the program is a set of instructions for
the pin to tell it “what we want it to do.” Ben continued to call learners’ attention to the pin that will be controlled as he explained the program on the shared screen. Slowing down his speech, in a low tone, he explained the program as "_telling_ *pin 10* to be high and _telling_ *pin 10* to be low" (57:25). Further, Ben drew students’ attention to the corresponding lines of code by hovering the cursor on the shared screen, tracing the sequential control flow during program execution.

Ben layered his explanations with voice modulation as he communicated the connections between power, circuit board, lights on the wristband, and the program. Building on a student’s question and response, Ben added that the program "*allows the electrons to _flow through_*" when pin 10 is set to HIGH. Ben completed the explanation, layering it with gestures in the air, tracing imaginary alligator clips and wristbands through which electrons would flow (see images and transcript in Fig. 10). Tracing a path using his right index finger, Ben said: "they flow through the alligator clips, which now flows to your *circuit* on your *wristband*" (1:05:54). He then asked the class what setting the pin to LOW would mean. Ben called it a closing gate and stressed how "it is *literally* just a *switch* but now it’s programmable." He expanded his explanation by layering gestures to enact pin 10 as a gate and the program as controlling the opening and closing of the gate, thereby controlling the electron flow. He layered gestures with voice to tell how gates, when open, "*allow* all the electrons to flood through the gate in which case the LEDs will turn on" or we could "*close* the gate and now the electrons aren't flowing through, and the LEDs turn off." Overall, communicating key aspects of the underlying processes involved not just verbal explanations and programming environment, but active embodied gestures, on-screen gestures, and voice modulation.
Figure 10: The multilayered enactment programming as controlling electron flow.

Ben: So, these *pins* are going to be your power. And, we are going to tell *pin 10* what we *want it to do*...
...I am _telling_ *pin 10* to be high_ and _telling_ *pin 10* to be low

Dave: Where are we getting the power from? Is there a battery in the circuit board?

Ben: How are the LEDs on the bracelet turning on? What is happening to the electrons... how are the electrons being manipulated to do this?... What did we *set* high and low? Did we actually turn the LED on and off or was it _something else_?

a. Ben: The program *allows the electrons to flow through*... when pin 10 is set to high.

b. Ben: They flow through the alligator clips, which now flows to your *circuit* on your *wristband*... *it is* literally* just a *switch*... but now its programmable.

c. d. Ben: Gates, when open, *allow* all the electrons to flood through the gate in which case the LEDs will turn on or

e. f. we could *close* the gate and now the electrons aren’t flowing through and the LEDs turn off.

Playing out Program Execution: Program as a Play

The focus of this lesson was to understand how a static piece of code will cause light patterns based on digital input values and _if_-conditional statement execution (shaded blocks on 031821 in Fig. 9; Fig. 11 to follow the episode). Ben set up to roleplay code execution of a simple program (c. and d. in Fig. 11) by stating roles and responsibilities: he set himself as the microcontroller that will follow the program line-by-line. He made most students take the role of lights, who will act according to his instructions. Ben set up the class teaching assistant as the digital input device (the switch) and another volunteer student as the variable that will store the switch state. Ben emphasized that the student would read the value only when asked to, just like lights would change their actions based on his instructions.

He adopted embodied gestures to communicate different types of ideas during this episode. At the outset, he concretized abstract program tokens _HIGH_ and _LOW_ by establishing hand gestures thumbs up and down to indicate them (a. and b. on Fig. 11). Starting to act as the
microcontroller, he read out the code line-by-line. He pointed to his forehead at the lines with variables to highlight how the microcontroller will remember variables for the rest of the program execution time (c. and e. on Fig. 11). He further pointed different directions in the air to show how the microcontroller would "activate" pins inside the setup() section.

Not only in gestures, but these ideas were also further communicated by his on-screen representations, particularly as he highlighted associated lines of code on the shared screen for the class to follow the control flow during program execution. As visible in Fig. 11, highlighting lines of code while roleplaying as a processor, he deepened his explanation to bring to life the sequential control flow of the program. Unlike the computer, which runs programs at an unperceivable pace, Ben's demonstration brought down the code execution speed for novices to understand the internal dynamics such as storing and updating values during program execution.

Voice modulation was the most prominent form of emphasizing specific ideas while ignoring others when Ben executed the loop() section of the program. In addition to continuous manipulation of on-screen representation and gestures, Ben changed his voice to personify the microprocessor, stress certain critical parts of the code, and slow down at selected phrases to draw further attention to action during runtime. As seen in the dialogues below, Ben changed his voice to highlight the microprocessor's personified role while emphasizing key parts of the program by modulating voice to stress on them. For instance, Ben turned his voice to be mechanistic while reading specific lines of code such as "~digital read switchy~" (d. on Fig 11) or "~if switch state is high, wave~". While running the loop() section a few times to highlight the repetitive nature of this module's execution, Ben constantly stressed the value he was reading from the switchState variable to stress further how that determines the conditional control flow during program runtime.
Implications

Analysis of Ben’s instructional work within this computing class calls for an expansion of our understanding of multimodal teacher enactments. Ben’s extensive adoption of embodied gestures aligns well with previous work (Kwah, 2013; Solomon et al., 2020) that has noted that teachers communicate metaphorical ideas and concretize abstract meanings in programs and invisible processes during program execution through gestures. More importantly, his layering of voice and its modulation to signal perspective change, emphasize key ideas, and draw attention by slowing down begs further exploration of how teachers within classrooms work with voice to communicate disciplinary ideas. Further, the context of online teaching sheds light on on-screen representations and manipulations, significant especially with enlarged shared screens visible for the whole class.
CHAPTER 4

“How does the computer carry out digitalRead()?” Learner conceptual agency while interacting with notional machines

This chapter will be published within ACM Digital Library.


Chapter Summary

Learners in computing classrooms must develop and adopt viable conceptions of program dynamics to read, write, and debug programs within a given programming context. Learners find support in notional machines, or simplified notions about underlying program behavior within a programming context, that educators provide them. While prior studies have examined students’ use of notional machines to trace program execution during interviews or quizzes, learners’ interactions with them within classrooms is understudied. Framing learning with notional machines as a socio-cultural process, I conducted an interaction analysis of high school students’ learning across eight class periods during a 14-week introductory computing electronic textiles online unit to answer the questions: (1) How did students interact with program dynamics during the unit? (2) Did notional machines support students with computing conceptual agency? If so, how? Findings revealed that learners interacted with program dynamics in agentic ways as they adopted notional machines to reason, ask questions about, reveal and revise their notions of program execution. While prior studies have explored notional machines’ affordance for mental model development, this study revealed their situated nature and their role in making program dynamics visible and accessible for sustained joint exploration within classrooms. Findings from this analysis make a three-fold contribution: they highlight the mediating role of notional machines in supporting agentic learner interaction with program dynamics, their role in sustaining joint
meaning-making within computing classrooms, and their evolving nature as a result of continuous interactions between the learners and the teacher.

Introduction

One of the primary goals of computing classrooms is to support learners to develop and adopt viable conceptions of program dynamics to read, write, and debug programs within a given programming context. Notional machines, defined as simplified notions about program dynamics that educators provide learners while teaching, are central to supporting learners within computing classrooms (e.g. Fincher et al., 2020; Guzdial et al., 2019). While studies have examined learners’ uptake of teachers’ notional machines while solving programming problems (e.g., Tunnell-Wilson et al., 2018), very few have explored the process in which learners develop viable conceptions about program dynamics within classrooms (Fincher et al., 2020). In this study, I framed learning with notional machines as a sociocultural process (Hall & Greeno, 2008; Vygotsky, 1978) and analyzed student interactions with program dynamics and the role of notional machines within an introductory computing high school classroom during a 14-week electronic textiles (hereafter e-textiles) physical computing unit. This analysis, one of the first ones to examine the learning process with notional machines through sociocultural theories, highlighted the role of notional machines as conceptual tools that mediated learners’ agentic interactions with program dynamics. Learners made visible their understanding of program dynamics and revised them while adopting notional machines and asking clarifying questions, overall participating as meaning-making agents in disciplinary practices such as reading, writing, and debugging programs throughout the unit.

This study was motivated by how beginner programmers struggle with naive conceptions about program execution while reading, writing, or debugging computer programs (duBoulay et al., 1981). From associating computers with human-like reasoning abilities (Pea, 1986) to applying mathematical and English-language semantics (Qian & Lehman, 2018), various informal
conceptions shape beginners’ sense-making of programs. Notional machines, such as representation of variables as boxes storing values or as parking lot spots, exemplify the “metaphorical layers of meanings” (Berry, 2015, p. 2) that teachers provide to support learners with approximations of the invisible processes underlying program behavior (Dickson et al., 2022; Fincher et al., 2020). In studies that have intentionally integrated notional machines, learners have affirmed their utility in understanding program dynamics (e.g. Dickson et al., 2022; Tunnell-Wilson et al., 2018). Further, teachers have also reported observing deeper understanding of programs among students when they taught with notional machines (Cutts, Robertson, Donaldson, & O’Donnell 2017).

Nevertheless, we have far fewer accounts of how students learn with notional machines within classrooms, particularly making sense of program dynamics while engaging in disciplinary activities like comprehending, writing, and debugging programs. Rooted in psychological and cognitive theories of learning, most research on notional machines has examined student learning in limited ways (Sorva, 2013; Shapiro in Guzdial et al., 2019). Earlier studies have framed student learning in terms of cognitive processes such as correcting misconceptions or developing mental models (Sorva, 2013). As a result, the majority of studies have captured individual student responses to think-aloud interviews and quizzes to analyze their mental models of program execution (Cunningham et al., 2017; Tunnell-Wilson et al., 2018). Although student responses to these tasks capture learner conceptions, they provide only a momentary snapshot of student learning, as opposed to deeper investigations of the processes culminating in learning. Do learners absorb and replicate notional machines that teachers present or do they actively engage with them by adapting them for problem-solving are open questions. This gap is significant and needs attention since notional machines are closely related to teaching-learning contexts and the meanings teachers and learners make of program dynamics (Dickson et al., 2022; Fincher et al., 2020).

Sociocultural learning theories call for an investigation of the process of learning to understand conceptual learning (e.g., Vygotsky, 1987). Learning, defined as learners’
“participation in the activities of communities of practice,” involves social interactions mediated by specific tools within a disciplinary community (Hall & Greeno, 2008). Of significance are learners’ interactions with aspects of the learning environment such as language and other conceptual tools that shape these actions, illuminating the two-way interactional relationship between the learners and the tools in the space (Vygotsky, 1978). Considering this way, learner actions are anything but isolated from context, and its cultural and historical practices. Learners, over time, not just develop conceptions but conceptual agency through which they feel “entitled and capable of questioning, criticizing, and adapting disciplinary resources rather than only using them mechanically” (Hall & Greeno, 2008; p. 18). Extending this understanding to computing classrooms, notional machines can serve as conceptual tools that mediate learners’ interactions with program dynamics while engaging in disciplinary practices like reading, writing, or debugging programs. Examining learning, then, centers learners’ computing conceptual agency “to appropriate, adapt, question, and modify conceptual meanings” in computing (Boaler, 2004; Hall & Greeno, 2008, p. 25). Although the computing education research community has hypothesized the mediating role of notional machines within computing classrooms (Fisler and Cunningham in Guzdial et al., 2019), no study so far has examined learning with notional machines as it takes place in interactions between teachers and students within classrooms.

In this paper, I draw on sociocultural theories of learning to examine computing learning within a high school class to answer the questions: (1) How did students interact with program dynamics during the unit? (2) Did notional machines support students with computing conceptual agency? If so, how? I report findings from interaction analysis (Erickson, 2006; Jordan & Henderson, 1995) of online class recordings (Angelillo et al., 2008; Derry et al., 2010) and student-generated artifacts from a 14-week e-textiles unit (Kafai et al., 2019) within the year-long introductory high school Exploring Computer Science curriculum (Goode, Chapman, & Margolis, 2012). An experienced computing teacher taught the unit with a focus on supporting learners with notional machines. Findings from the analysis illuminated the agentic role learners took on while interacting with program dynamics and the mediating role of notional machines. Learners adopted
notional machines discussed in the class, asked clarifying questions, made their understanding of program dynamics visible and revised them, and roleplayed program execution. Across the accounts, the situated nature of notional machines, their role in sustaining learners’ meaning-making of program dynamics during a unit, and the contribution of sociocultural theories for future notional machine design and research were evident.

Background

Learning with Notional Machines

Teachers simplifying disciplinary ideas to make them accessible for novices have a long legacy within science and mathematics education. By providing learners with analogies and metaphors, science and mathematics teachers have supported learners to understand the invisible processes underlying the phenomenon under investigation. The ability of metaphors, for instance, to “highlight and hide” aspects of the phenomenon (Lakoff & Johnson, 2008) allows teachers to tailor complex ideas based on learners’ prior engagement within the discipline. Simplified explanations such as electric current as a “flowing fluid” have aided high school student learning in science classrooms (Gentner & Gentner, 1983). Drawing analogies between arithmetic operations and manipulating blocks, teachers have supported elementary school students learning abstract place value concepts (Bowers, Cobb, & McClain, 1999). Similar efforts within computing classrooms take the form of notional machines where computing processes are simplified to support learners while drawing their attention to only particular aspects of the process (Fincher et al., 2020).

Reviews conducted by Sorva (2013) and, more recently, Fincher and colleagues (2020) highlight key characteristics of how researchers have conceived learning with notional machines so far. As Sorva (2013) points out, the majority of efforts in understanding learners’ engagement with notional machines have focused on capturing learners’ conceptions, misconceptions, and mental models of program dynamics within a given programming context. Fincher and colleagues
(2020) have captured the same trend in their visualization of how learners learn with notional machines. As shown in Fig. 12 (left), notional machines, thought of as “a-kind-of” conceptual model of program dynamics with pedagogical purposes, are created by teachers, and individual learners make a “personal version” of these machines in the form of mental models. This approach is comparable to the acquisition model of learning (Kafai et al., 2019; Sfard, 1998), and studies adopting this approach focus on assessing whether learners acquired particular models of program dynamics. For instance, Tunnell-Wilson and colleagues (2018) examined student responses to program evaluation quizzes to understand students’ problem solving with a substitution notional machine, their misconceptions, and tendencies to avoid certain parts of the model while working with recursion. Along similar lines, Cunningham and colleagues (2017) analyzed student sketches of control and data flow to understand their conceptions of program execution. Although these studies shed light on student conceptions of program dynamics, they capture the end products of learning rather than the process (Bowers et al., 1999). As modeled in Fincher and colleagues’ (2020) depiction, the research community has conceived a unidirectional arrow from notional machine to the learner that precludes any interaction between learners and notional machines during the learning process other than students receiving teachers’ notions. However, as Dickson and colleagues (2022) also point out, learners do not merely “passively observe” (p. 852) notional machines but can interact with them by adopting them and asking questions about them while developing disciplinary agency within computing classrooms.

Figure 12: Fincher and colleagues’ (2020) model for teaching and learning with notional (left); revised model based on sociocultural theories of learning (right).
Viewing learning with notional machines through sociocultural theories calls for an attention to the process of learning rather than the end product alone (Vygotsky, 1978). Sociocultural theories posit that learning happens in interactions as learners participate in disciplinary practices, adopting tools within the community (Hall & Greeno, 2008). For instance, third-grade students in a mathematics class engaged in communal mathematical practices of performing arithmetic operations as they made sense of place values (Bowers et al., 1999). Here, the teacher provided the class with a simplified “model of” performing place value transformations, which then students made into a “model for” performing computations while adding and subtracting numbers. Along similar lines, computing classrooms can be viewed as the socialization of communities of learners to disciplinary practices such as program comprehension, debugging, and program writing (Brennan & Resnick, 2012; Grover & Pea, 2013). Tools such as programming languages, environments, interfaces, and visualizations, designed for pedagogical purposes, mediate this activity as learners make sense of program dynamics (Sorva et al., 2013). Notional machines can be treated as similar, but conceptual, tools within computing classrooms that mediate learners’ meaning-making of program dynamics within a programming context (Guzdial et al., 2019). In that case, understanding learning with notional machines should include an examination of learner interactions with these tools to capture ways in which they learn about program dynamics and how they participate in disciplinary practices. To further build this argument, I will briefly describe parts of sociocultural theories that provide a framework to examine learning with notional machines.

Sociocultural Theories of Learning

A call to focus on the social, cultural, and historical nature of learning in the 1970s came as a dissent from psychological theories of teaching and learning (Roth & Lee, 2007; Vygotsky, 1978). Sociocultural theory highlighted the context and viewed learners as situated within communities with shared meanings and practices and learning as socially situated and constructed in interactions. According to sociocultural theories, learner communities engage in “regular and
recurring patterns of activity, called practices” (p. 128; Greeno & Engerström, 2014), through the adoption of shared tools (Vygotsky, 1978). Teachers provide learners with simplified yet functional notions of more sophisticated formal concepts for them to engage within disciplinary practices (Greeno, 2012). Learners’ sense-making moves from the social to the individual plane, “mediated” by these conceptual tools, “broadens the range of activities” in which learners can participate (Vygotsky, 1978, p. 55). Over time, learners develop shared meanings and interpretations, and agentic ways of interacting with the discipline (Hall & Greeno, 2008; Engerström, 1999; Shapiro in Guzdial et al., 2019). Examining learning from this perspective implies exploring learners’ conceptual agency, defined as ways in which learners adopt, critique, and reorganize concepts to problem-solve within the discipline (Boaler, 2002; Hall & Greeno, 2008).

Sociocultural theories have been adopted to study student interactions within classrooms in other disciplines, particularly mathematics. For instance, Bowers and colleagues (1999) studied how third-grade learners learned arithmetic with functional concepts of “packing and unpacking” of cubes. Here, the teacher presented “packing and unpacking” as a simplified notion of formal place value concepts and scaffolded learners to perform arithmetic operations. The learners adopted the teacher’s simplified conception of packing and unpacking as a tool for problem-solving while engaging with mathematics in agentic ways. Similarly, Jurow (2004) examined mathematics learners as they engaged in curricular activities to understand how they developed conceptions of generalization. A comparable analysis in another mathematical activity (Ma, 2016) demonstrated the different ways in which learners recruited resources and enacted conceptual agency while participating in geometry activities. Sociocultural theory has led to the exploration of the students’ mathematical sense-making by drawing attention to the conceptual tools mediating learning. However, despite potential affordances for holistic consideration of teaching and learning within classroom settings, this perspective has hardly been explored in computing contexts.
Computing classrooms, like other disciplinary learning settings, can benefit from sociocultural perspectives (Ben-Ari, 2004; Kafai et al., 2019; Shapiro in Guzdial et al., 2019). Viewing computing teaching and learning through sociocultural lenses, notional machines are conceptual tools provided to novice programmers in computing classrooms to make sense of program dynamics in a programming context (consisting of language, paradigm, and tools, at the least). Formal semantics is one such symbolic system that can serve as tools within advanced post-secondary computing classrooms or expert computing communities with “high mathematical sophistication” (Guzdial et al., 2019, p. 2). However, when such a shared symbolic system is missing within introductory computing courses, teachers resort to notional machines, more accessible and simplified tools, to communicate and discuss program dynamics. For instance, the notion of “variables as parking spaces” (Fincher et al., 2020) can allow a teacher to communicate the relationship between the type and the value of the variable. Although we have no accounts from classrooms, such notions can serve as functional concepts for learners to engage with program dynamics during some parts or throughout the unit (Greeno, 2012).

While students learn with functional concepts, they are also expected to adopt these notions while participating in disciplinary activities, such as comprehending, writing, or debugging programs (duBoulay et al., 1981). More importantly, they must demonstrate conceptual agency by navigating problem spaces while adopting and extending these notions (Boaler, 2004; Ma, 2016). For instance, do students adopt the notion of “variables as parking spaces” while comprehending, writing, or debugging code, is an open question. Or, what will learners’ “personal copies” of these notions look like, how long do they adopt them, or at what point in their learning do they question the limitations of these ideas are open for exploration. Framing learners as active meaning-making agents participating in activities allows for observing learner interactions with program dynamics as mediated by notional machines within classrooms. Individual students may not only receive teacher’s notional machines but can demonstrate their developing conceptual agency by adopting notional machines in different ways while reading, writing, and debugging programs. However, almost no study has examined learning in classrooms where
learners interact with teachers, peers, and concepts such as notional machines (Fincher et al., 2020), and even fewer have explored learning in terms of students’ agentic interactions with disciplinary ideas within computing classrooms.

Different programming environments, languages, and associated paradigms offer diverse contexts within computing classrooms to explore notional machines (Krishnamurthi & Fisler, 2019). For instance, learning with physical computing construction kits such as electronic textiles (e-textiles), for example, involve physical circuits in addition to on-screen programs (e.g., Jayathirtha & Kafai, 2020). Learning computing within this context includes learning the underlying behaviors of programs and their interactions with circuits during execution. However, students have trouble learning computing with physical computing, particularly understanding the interaction between programs and circuits. For example, DesPortes and DiSalvo (2019) noted novice undergraduate students’ challenges while comprehending a simple Arduino program causing two LEDs to blink. Students’ difficulties understanding the semantics of function calls came in their way of reasoning sequential program execution. Similar observations were noted in Booth and Stumpf’s (2013) study. Students, even with prior programming experience, had challenges grasping program execution within a physical computing programming environment, primarily when bugs were distributed between circuits and programs (Jayathirtha & Kafai, 2021b).

In sum, learners will need support to understand the interaction between programs and circuits to be able to reason, write, and debug programs. Teaching with physical computing systems requires teachers to go beyond the visible components and support learners to understand internal behavioral details that can help reason interactions during program execution, such as data and control flow (Ananthanarayan & Boll, 2020). For example, e-textiles hide the internal processes such as how variables get created or how values are stored and manipulated during execution that challenge students’ understanding of the overall system behavior (Jayathirtha et al., 2018; Jayathirtha & Kafai, 2021b). However, we have no accounts of any notional machines that may be adopted while teaching within these programming contexts (Krishnamurthi & Fisler, 2019).
Overall, as seen in Fig. 12 (right), viewing learning with notional machines through sociocultural theories: (1) notional machines can serve as functional concepts within computing classrooms, (2) learners learn as a community while engaging in practices like comprehending and debugging programs, (3) learners make meanings about program dynamics while interacting with the teacher and peers, and (4) agentically interact with program dynamics, potentially adopting and questioning notional machines. Learning with notional machines, framed as a sociocultural process, guided the data collection and the analysis in this study.

Methodology

School and Curricular Context

The study was conducted at a public charter high school in a U.S. West Coast city, in which nearly 75% of students identify as non-White and 52% opted for the free or reduced lunch program during the study. The school offers a computing pipeline—i.e., a series of courses that students can take between their freshman and senior years, from introductory Exploring Computer Science (ECS) to Advanced Placement Computer Science courses. Ben, the participant teacher in the study, designed the pipeline during his tenure as Chair of the Computer Science Department. He was teaching for his thirteenth year at the school during the academic year 2020–21.

ECS was offered as an introductory course within the school. Students, mostly freshmen, without formal programming background took the course. The year-long curriculum was designed along three strands: concepts, inquiry, and equity (Goode, Chapman, & Margolis, 2010). Ben taught a revised version of the e-textiles unit as a culminating unit within ECS during Spring 2021; this was his sixth year teaching the unit. The constructionist curricular unit was designed for students to make four fabric-based projects while exploring circuitry and computing ideas (Kafai et al., 2019; see Fig. 13, below for a sample project). Ben used the Adafruit Circuit Playground microcontroller and other sewable components such as lights and conductive thread while
implementing this unit (see Fig. 13 below). Between projects, specific lessons supported learners with additional context or computing concepts such as programming with lights and digital inputs, in order to enable them to make their own functional and personally meaningful projects.

Regarding the e-textile programming context, Ben used the Adafruit Circuit Playground microcontroller and sewable electronic components such as lights and conductive thread during this unit (see Fig. 13, above). The class programmed the microcontroller using the text-based Arduino programming language and followed an imperative programming paradigm. Programs in the unit mostly consisted of three parts: global variable declaration and/or definitions, setup() function to assign microcontroller pins specific roles during program execution, and loop() function to program the expected outcome on the circuit, comparable to the main() function in other programming languages except that this repeatedly loops during execution (see Karla’s project code in Fig. 13, below). The class used a simple online on-screen programming environment that offered only a code editor and a serial monitor to view data streams whenever needed. Program outcomes such as light patterns were visible on accompanying physical circuits (see Fig. 13, below).
During this study, Ben taught a version of the e-textiles unit that was adopted to accommodate the online teaching and learning context due to the COVID-19 pandemic and also to focus on teaching program dynamics using notional machines (see Table 5). During Fall 2020, Ben collaborated with the research team to redesign the curriculum for online implementation based on his experiences teaching online unprepared during Spring 2020 (Jayathirtha et al., 2020). In addition, Ben and I met six times (~1 hour each time) to discuss student challenges learning with e-textiles based on our joint experiences and the potential use of notional machines to mitigate them. We identified some notional machines that Ben had already used in previous years teaching the e-textiles unit (e.g., programs as plays and roleplaying program execution while introducing new programming constructs). We also identified parts of the curriculum where students could benefit from simplified notions of the underlying mechanics (e.g., circuit behavior, relationship between circuits and programs during runtime). Overall, the agenda was not to
design notional machines for the unit but to orient the unit towards supporting learners to develop conceptions of underlying program dynamics within e-textiles and of providing learners with opportunities to participate in reading, writing, and debugging of programs during the unit.

During Spring 2021, he taught the curricular unit online. The class met on the Zoom online meeting platform on alternate days (~80 minutes each day) for 14 weeks. Throughout the unit, Ben multiple notional machines spread across five themes to make program dynamics accessible to novice learners (see Table 6). They differed in granularity, i.e., they supported learners in understanding either individual symbols, statements, blocks of programs or entire programs. Further, they differed in forms, i.e., simple explanations, analogies, short skits, or elaborate roleplays. In sum, they were intended to support the class to engage with program dynamics throughout the unit.

Of particular interest for the accounts shared in the findings below are three specific notional machines: variables as “remembering” values, digitalRead() statement execution as “reading” values, and program execution as plays. Variable names were introduced and discussed as labels to remember values during program execution, just as humans need names to remember people. Further, execution of functions such as digitalRead() and digitalWrite() were also discussed in terms of anthropomorphized microcontrollers. Microcontrollers reading and writing electric signals at their pins during these function executions were compared to humans reading and writing information. Further, to strengthen connections between program execution and the circuit-level outcomes, the class discussed program execution as a roleplay. According to this notion, programs acted as scripts for the play, and microcontroller as the “director” of the play who would instruct the lights to perform actions or reads from switches, remember and update the value, and perform conditional logic. Overall, we conjunctured that notional machines would provide learners with notions of certain underlying behaviors.
Table 6: Projects, lessons, and notional machines within the revised unit taught by Ben.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual Focus</td>
<td>Simple circuit</td>
<td>Maker ethos</td>
<td>Parallel circuits</td>
<td>- Programming light patterns</td>
<td>- Programming with two digital input buttons</td>
<td>- Debugging at the intersection of circuits and programs</td>
<td>- Programming with analog sensors like light and sound sensors</td>
</tr>
<tr>
<td>Notional Machines</td>
<td>Circuit as electron loops</td>
<td>--</td>
<td>-Circuits as electron loops - Switches as gates</td>
<td>-Circuits as electron loops - Microcontroller pins as programmable gates - Programs as plays - Anthropomorphized computer and microcontroller</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Notional machine themes, their granularity, and their forms.

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Themes ↓</th>
<th>Individual Symbols</th>
<th>Individual statements</th>
<th>Blocks</th>
<th>Entire Program</th>
<th>Circuit only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthropomorphized Computer</td>
<td>=, { }, ;</td>
<td>int led = 10; as “led gets the value of 10” as “understood” by the computer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programs as Plays</td>
<td>Variables as character names</td>
<td>setup() as characterizing</td>
<td>Execution as play</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anthropomorphized microcontroller</td>
<td>==, &amp;&amp; as “check if” and “and” in English</td>
<td>digitalRead(); delay(); as reading and causing delays</td>
<td>loop() if-blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microcontroller pins as programmable gates</td>
<td>pinMode(); digitalWrite();</td>
<td></td>
<td>Simple blinking light program as manipulation of gates</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Circuit as Electron Loops

| How statements like digitalWrite(); cause electron-level changes |
|---|---|---|
| How do lights turn on? What does power mean? |

Yellow: Explanations; Purple: Analogies; Green: Short skits; Blue: Roleplays

**Participants**

The teacher, Ben, chosen for this case study, had had research-based relationships with my research team for over 6 years. His extensive teaching experience, particularly the e-textiles unit within ECS, made his class a suitable site for an exploratory study about notional machines within high school classrooms. After being introduced to notional machines during Fall 2020, Ben chose to integrate them with ECS objectives to make concepts accessible to learners in equitable ways. I took the role of participant-researcher (Erickson, 1985) and took the role of a teaching assistant (TA) during the unit.

Twenty-four of the 34 students and their parents had consented to participate in the research study. Among them, pseudonyms were assigned to students who consistently participated in the class in visible ways by unmuting themselves or typing in the chat box in response to teacher queries or turning their cameras on during synchronous online sessions. Mona and Karla identified as non-White females, Juan, Harem, and Adyan identified as non-White males, and Dave as a mixed-race male (all pseudonyms). The online setting made it almost impossible to capture process data of students who turned off their video feeds and were unresponsive in the chat, although they regularly shared the progress of their projects (e.g., versions of their code, aesthetic and circuit designs of the projects) through an online app, Seesaw, that was used in the unit.
Data Collection and Analysis

Data from two sources were analyzed to examine student interactions with notional machines during the unit: recorded class videos of online sessions and end-of-lesson student reflections. I screen-recorded all the 37 class periods (80 min. each) and wrote end-of-day summaries throughout the 14-week unit. The class met every alternate day on the Zoom online platform. Screen recording captured the chat window, learner and teacher windows, and any screen shares on Zoom. Students shared their reflections on Seesaw, an online app that was used in the class to support peer sharing.

Inspired by interaction analysis (Erickson, 2006; Jordan & Henderson, 1995), I analyzed a select set of classroom recordings (Angelillo et al., 2008; Derry et al., 2010) while constantly paying attention to the process of learning and generating explanations (Vygotsky, 1978). Previous studies exploring student agentic participation within classroom activities have been guided by sociocultural theories and interaction analyses methodologies (e.g., Bowers et al., 1999; Erickson, 2006; Jurow, 2004; Ma, 2016). Below, I explain the data analysis process stepwise for readability, although the overall process was not as linear. To make methodology and resultant findings recoverable (Checkland & Holwell, 1997), I share explicit theoretical mappings and examples whenever helpful. Moving from whole-to-parts (Erickson, 2006), I describe steps starting from all class videos and ending with a set of exemplary episodes to highlight student agentic interactions with program dynamics.

Step I Selection of class periods for analysis

I read all the class period summaries to select days when Ben explicitly invited students to engage with program dynamics (8 out of 37; see Table 7). On these days, Ben requested students to do one of the following with programs: co-authoring, reasoning and predicting outcomes, debugging, or roleplaying execution. The rest of the days, the class worked with
circuits or designed their e-textiles projects in small groups. Any discussion about programs on these days focused on their functionalities or outcomes, with almost no mention of underlying behavior.

**Step II Generating content logs**

For each of these eight selected days (dates written in MMDDYY format), I watched the videos repeatedly to generate content logs (Jordan & Henderson, 1995) with student and teacher utterances, gestures, chat entries, and shared screens and/or documents related to underlying processes transforming programs to outcomes. I appended logs with additional data from their Seesaw entries like end of the day student reflections or codes whenever available. Names and appearances of the teacher and students were anonymized across the text and screenshots of the video.

**Step III Activity as a unit of analysis**

For each period, I segmented (Jordan & Henderson, 1995) content logs to focus on program-dynamics-related activities that the class as a community engaged in. This led to the identification of thirteen activities across eight class periods (see Table 7 for the list of all activities). Each activity usually opened with an investigative question, followed by joint exploration and a summary of the answer to the question. Some periods had one such activity (e.g., jointly writing a program throughout the class period) and some had a series of activities (e.g., predicting the outcome and roleplaying program execution). Activities where program-dynamics–related engagement was not visible such as learners programming light patterns (e.g., 031221) were not considered for further analysis. Activities varied in duration, anywhere between 5 and 28 minutes.
Table 8: Dates and activities involving student engagement with program dynamics.

<table>
<thead>
<tr>
<th>#</th>
<th>Date (MMDDYY)</th>
<th>Activity (Duration in mins)</th>
<th>#</th>
<th>Date (MMDDYY)</th>
<th>Activity (Duration in mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>031021</td>
<td>Reasoning observed outcome (blinking lights) in relation to the program (~10)</td>
<td>8</td>
<td>041621</td>
<td>Writing &amp; reasoning a program with two digital button inputs (~9)</td>
</tr>
<tr>
<td>2</td>
<td>031221</td>
<td>Debugging the program to make lights blink and Reasoning the same (~7)</td>
<td>9</td>
<td></td>
<td>Reasoning program execution &amp; Predicting circuit outcome in relation to program execution (~7)</td>
</tr>
<tr>
<td>3</td>
<td>031621</td>
<td>Predicting Program outcome i.e., particular light pattern (~6)</td>
<td>10</td>
<td></td>
<td>Predicting circuit outcome in relation to program execution (~5)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Roleplaying Program execution and Predicting the outcome (~12)</td>
<td>11</td>
<td></td>
<td>Roleplaying Program execution (~5)</td>
</tr>
<tr>
<td>5</td>
<td>031821</td>
<td>Writing and Reasoning a program with a digital input switch (~28)</td>
<td>12</td>
<td>042221</td>
<td>Debugging &amp; Reasoning &amp; Roleplaying a program with two digital button inputs (~14)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Roleplaying Program execution (~4)</td>
<td>13</td>
<td>052021</td>
<td>Writing and reasoning a program with an analog sensor (~10)</td>
</tr>
<tr>
<td>7</td>
<td>041421</td>
<td>Debugging and Reasoning a program with two digital button inputs (~8)</td>
<td></td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>

---

**Step IV Analyzing activities for student actions**

Guided by sociocultural theories of learning, analysis of each activity started from the social plane and moved towards individual agents and their participation (Vygotsky, 1978). I generated descriptive vignettes for each activity and analyzed them for student actions while making sense of programs. For instance, the first activity, reasoning through the behavior underlying a blinking light (031021), was described to capture the context of the class as programming its first light pattern. I included details such as notional machines discussed so far within the unit. Within this
context, individual student interactions were further analyzed. For instance, I noted student actions such as Dave asking a clarifying question “where are [LEDs] getting the power from?” or Juan contributed by adopting an notional machine discussed earlier in the context of simple circuits.

Explanations were generated for every action by tracing any histories from previous class activities, thereby keeping student action embedded within the context of the activity and allowing for any mediational role of notional machines to surface (Erickson, 2006; Vygotsky, 1978). For example, Juan’s description of the digitalWrite(led, LOW); statement as “it’s closing a gate” (031021) was explained as Juan adopting a previously discussed notional machine. The class had discussed switches in simple circuits, during the first two curricular projects, as “gates” controlling current flow within the circuit. Juan, by comparing digitalWrite() statement execution to a gate, adopted that notion and explained the interaction between the program statement and the microcontroller pin as a “gate controlling the electron flow.” This way, student interactions with program dynamics and the potential role of notional machines were situated within the context of classroom discussions. Data throughout the unit was accessed iteratively to generate explanations that captured the contextual meanings of the interactions. Reliability of explanations as possible interpretations was established in a collaborative fashion (McDonald et al., 2019). Explanations for all fifteen activities were generated in iterative passes while constantly consulting with four other computing education researchers with expertise in sociocultural theories. Data and its accompanying explanations, just as shared below, were presented to the experts to generate alternative interpretations and develop holistic explanations (Jordan & Henderson, 1995).

Step V Developing analytic categories

I repeatedly read and inductively analyzed activities and accompanying explanations to generate themes that capture various ways in which students engaged with program dynamics (Jordan & Henderson, 1995; Creswell & Poth, 2016). Since the focus of this paper is to highlight the complexity of student interactions with program dynamics and the role of notional machines in
shaping them, emphasis was placed on establishing reliability for the interpretations rather than the emergent themes (McDonald et al., 2019). Thus, findings below are structured as detailed accounts of classroom activities in which students engaged in disciplinary practices such as reading and writing code. Contextual descriptions shed light on these connections. Below I report exemplary accounts to highlight how learners engaged with program dynamics and the role of notional machines in shaping it.

**Table 9: Analytic and operational categories.**

<table>
<thead>
<tr>
<th>Codes</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Questioning</td>
<td>For clarifications</td>
</tr>
<tr>
<td>Adopting</td>
<td>Prior non-computing experiences</td>
</tr>
<tr>
<td></td>
<td>Prior computing experiences</td>
</tr>
<tr>
<td></td>
<td>Prior notional machines in the unit</td>
</tr>
<tr>
<td></td>
<td>Revised notions</td>
</tr>
<tr>
<td>Reasoning</td>
<td>Circuit behavior in relation to program execution and visa versa</td>
</tr>
<tr>
<td>Roleplaying</td>
<td>Contributing to setting up the roleplay</td>
</tr>
<tr>
<td></td>
<td>Enacting predicted outcomes</td>
</tr>
<tr>
<td></td>
<td>Enacting program execution</td>
</tr>
<tr>
<td></td>
<td>Reflecting on the roleplay</td>
</tr>
<tr>
<td>Confirming</td>
<td>Use of an analogy</td>
</tr>
<tr>
<td>Humoring</td>
<td>Alternative meanings to tokens</td>
</tr>
<tr>
<td></td>
<td>Surfacing tensions in anthropomorphized notional machines</td>
</tr>
</tbody>
</table>

**Step VI Selecting activities for reporting**

I selected four activities across two days of the unit, almost a month apart (031821 and 041621), to elaborate on students’ interactions with program dynamics and the role of notional machines. The selected two days together included the most common ways in which students interacted with program dynamics by reasoning, questioning, and roleplaying program dynamics. These activities, mid-way in the unit, illuminated the role of notional machines discussed until then in shaping student interactions with program dynamics.
I shortened the activities to make them easier to read by focusing on students’ interactions and any notional machines enacted. I italicized words or phrases that Ben stressed on while teaching in addition to including screenshots to highlight the multimodal nature of communication within the online teaching-learning context. In addition, the first activity (031821) was fragmented into four episodes that together constituted writing and reasoning a program with a digital switch input.

Findings

In response to the first question of how students interacted with program dynamics during the unit, the analysis revealed that students demonstrated conceptual agency as they reasoned, questioned, roleplayed, and even humored while interacting with program dynamics (Table 8). And, regarding the second question about the role of notional machines in supporting learner conceptual agency, notional machines made visible and mediated their meaning-making of program dynamics. Below I share details about the relationship between student sense-making and notional machines as accounts of activities across two days during the unit.

In the first activity (031821), students contributed to writing a part of a program, reasoning about digital input processing for the first time during the unit and asked clarifying questions about the underlying behavior. In the second activity on the same day, learners roleplayed program execution and participated multimodally while revealing their conceptions of program dynamics and revising them. A month later (041621), students adopted previously developed conceptions to reason through programs and predict outcomes with two digital button inputs. Throughout, the historical nature of meaning-making with notional machines was evident as students constantly drew from prior classroom discussions to make sense of program dynamics during the unit. Further, despite the simplified approximations of actual program dynamics provided by the notional machines, they supported and sustained novice learners’ agentic interactions with program dynamics. Below I will provide some curricular context and activity accounts, each with a summary followed by interaction data and analytical explanations.
Curricular Context

By twelfth day in the unit (031821), the class had discussed variables, in a simplified way, as “remembering” values for future reference within a program (031021, 031221). Using an anthropomorphized computer notional machine, the underlying behavior of variables was compared to humans remembering information. However, all of these discussions involved variables to “remember” a fixed numerical value such as microcontroller pin number. The activity discussed below was the first time the class had to use a variable to store digital input values read from a switch. Ben hoped that the students adopted their conception about variables as mechanisms to “remember” values and extended it to this new program. While writing the program to read the digital input, Ben paused at the digitalWrite() line and invited students to complete the statement. He had written out the program with variables storing pin numbers (called as switchy in the program) and the setup() section. In preparation for roleplaying the program (in Activity 2), I was assigned to serve as the switchy with my phone light indicating the state of the digital switch. Turning on the light meant that the switch was on and turning off the light implied the switch was off.

Activity 1: Writing and Reasoning Program with a Switch

Summary

In this activity (031821), learners adopted previously discussed notional machine of variables as remembering values and reasoned through program execution. They revised their understanding and asked further clarifying questions about the dynamic behavior of programs within the e-textile context. Overall, learners were not just receiving the notional machines presented by the teacher but constantly made meaning with them and reorganized the disciplinary problem space as active agents.
Episode 1. Adopting prior notional machine discussed in the unit

48:14 Ben: How is our program going to remember what Switchy was? [a brief pause] Switchy is 21 but how are we going to remember the state of Switchy? We can read it, but…

Mona typed in “variable” in the chat

Ben: Oh, Mona says make a variable.

Analysis

Ben solicited the previously discussed notional machine by using the same language of variables as “remembering” values. Despite the new context of processing digital inputs, previous simplified discussions about variables as remembering values helped Mona to contribute to writing the program. This was seen in how she suggested Ben make a variable in order to “remember” the digital input value. Mona’s suggestion “variable” in the chat demonstrated her adoption even surprised Ben, as noted in his tone. Variables as “remembering” values served as a simplified yet functional concept for Mona to identify the need for a variable and meaningfully contribute to
completing the program statement. As Vygotsky (1978) noted, such simplified notion broadened the range of activities that learners like Mona could engage in i.e., by providing opportunities to contribute to writing a program.

Episode 2. Making naive conception visible and revising it

TS Transcript

Ben: What would you like to name that variable?
Dave types “On = Hokey Off = Pokey” in the chat
Ben: You know what, let’s talk about why we need a variable. Why do I need a variable?
[Ben and the TA set up for a short roleplay between themselves to highlight the underlying behavior; Ben acts as the processor that needs to read the switch in order to recognize the value and store it to remember]
TA: You are the processor and you don’t have eyes?
[Ben closes his eyes]
TA: I am the switchy, I am going on and off. How will the processor know if I am on or off? [Ben has his eyes closed]
Dave types: Because the computer needs to know the current state of the switch
Karla types: so you know in the code when to check?
Ben: I have to read you. I don’t know what you are right now.. [opens his eyes] you are currently off. My eyes are closed again.

TA: Now, the class wants you to turn on the lights if I was on and turn off the lights if I was off.
Ben: But, I don’t remember what you are.
TA: Why don’t you remember? You just read!
Ben: But, I didn’t store it in my memory. I am really dumb.
TA: We should then help you remember what you just read. How have we helped the processor remember in the past?
This is a question for the class.

51:37 TA: [reading from the chat] A student says that by making a variable, the CPU can remember what state the switch is at. Dave has typed “Can you set intervals for the CPU to check when switchy is on or off?”
Ben: Well, it’s in a loop(). So, it’s going to do this over and over. Because it’s a loop().

52:05 Ben: I don’t remember. I didn’t store the information. I don’t remember what it was. But, I read it. So, have any of you, like read a book, and you like got to two pages, and then you realize that you were reading but you had no idea what you were reading?
Nicole smiles and nods in agreement; Karla, Juan, Santos, Mona, and Nicole type in “yes,” “yup” in agreement.
Ben: So, that essentially is what a computer is always doing. The computer is always kinda existing but unless we tell it specifically to remember something, it’s not going to
remember it. We have to tell it to remember things! So, the fact that it’s reading the switch, that’s not helping us. We need a variable because we need to store what it read.

54:15 Ben: We need a name for this and this is remembering the current state of the switch or it’s storing its value. What do you want to name this variable?
Dave types store, Harem types “storage”

Analysis

This episode revealed the different ways in which students made meaning of program dynamics, constantly mediated by the same notional machine of variables as remembering or storing values. In this particular instance, this simplified notion brought program dynamics to the social plane and made the program accessible for the teaching team and the learners to discuss underlying invisible behaviors (Vygotsky, 1978). Learners, in interaction with the teacher and the program dynamics, both revealed and revised their conceptions, moving from the social to the individual plane.

Probing for a name for the variable revealed Dave’s naive conception that values cannot be overwritten in a variable. His suggestion to make two variables, one for every possible value read, led the teaching team to pause and share a couple of simple ways of making sense of variables and their behavior during runtime in the context of storing values from an input data stream. The teaching team, through a short skit and an analogy, approximated variables as remembering information (which is of significance in the following episode). At the end of the notional machine enactment, Dave revised his response and suggested “store” as the variable name. Suggestions made by other students also indicated their thinking along similar lines. Later in the unit (041821), Dave made two similar variables in his project to store two digital button inputs and named them “leftyState” and “rightyState,” further demonstrating his revised understanding.
Episode 3. Questioning for clarification

TS Transcript
1:02:32 Mona has typed in the chat that Ben reads out
Ben: “I know this is kind of off topic, but how does the sensor know if switchy is high or low?” Not off topic at all. What do you think the switch is doing? And what sensor? There’s no actual sensor there.

TA: Feel free to unmute yourself if that’s going to be easier.
[Mona unmutes]
Mona: Because I know we talked about how there was a sensor that senses if the switch is on or off. I was just wondering. I think we use in the code digitalRead(). So, how does the computer carry out the digitalRead()?
[Ben is busy drinking water]
TA: That’s a great question. So when you were doing digitalWrite(), can you tell what was actually happening? What was the computer doing when digitally writing?

Mona: It was turning on the LED or turning it off.
Ben: How was it turning on? What was it doing to the pin?
TA spreads her arms like releasing signals, a gesture the class has been used to by now to discuss microcontroller actions during runtime
Mona: It is increasing or decreasing the flow of electrons at the pin.
Ben: Yes, it was energizing the current. So, what do you think the switch is actually doing? [gestures physically turning on and off the switch]... We should say that there is a positive line going to pin 21 and there’s a ground coming from the switch. You just can’t see it. It’s built into the microcontroller.
Mona: Oh, then we are just measuring the flow of electrons in pin 21!
Ben: Yes! So, what do you think the switch is actually doing?
Mona: So, it’s basically just stopping or increasing the flow of electrons at pin 21.

Analysis

Learners initiated new lines of inquiry by asking clarifying questions while working with teachers’ notional machines. This was visible above when Mona asked about how the digitalRead() actually reads the input signal during runtime. As a result of the earlier short skit while discussing variables as remembering values, Mona thought that there was an intermediary sensor literally...
“reading” the switch and supplying the value to the variable, just as Ben “read” the state of the switch in the skit through his eyes. The lack of visible electric connection between the switch and the microcontroller board further challenged Mona’s understanding of the underlying mechanism while reading the switch. However, she demonstrated agency in sense-making of program behavior during runtime and asked a question that made her conception visible: “how does the computer carry out the digitalRead()?"

Investigations such as these usually made room for the teacher and the students to negotiate meanings, and if required, for the teacher to provide alternate explanations. In this case, Mona’s inquiry gave an opportunity for the teaching team to connect this question back to earlier conversations of how the digitalWrite() statements work (031221). Mona’s conception of “increasing and decreasing the flow of electrons at the pin,” although only a partially accurate approximation, served as a functional concept to engage with circuit behavior during program execution. Microcontroller pins as programmable gates notional machine, discussed in the previous class (031621), had involved discussing programs as manipulating the control flow at pins to cause blinking lights at the circuit. Ben, while clarifying Mona’s approximation as “energizing the current,” also pointed out the invisible connections between the microcontroller and the switch. Mediated by the conception that electric current varies at the microcontroller pins based on the program and the state of the circuit components, the teaching team and Mona engaged in an exchange that led Mona to revise her concept and understand the execution of the digitalRead() statement as “measuring the flow of electrons at pin 21” and that the switch will either “stop or increase the electron flow at pin 21.”
**Episode 4. Calling for revising the notional machine**

**TS Transcript**

1:06:51 Ben reads out Dave's question in the chat

Ben: “You said that there are electrons that are always present so how does the computer make them flow?”

Ben sighs!

Ben: I mean, can we just be okay with the fact that when there’s a power source, the power source energizes electrons and shoots them out? Kind of like.. Got it!! Remember those Hot Wheels that let you make like a loop with your Hot Wheels? There was like a spot where there were two wheels spinning and when Hot Wheels got to that spot, it would zoom ahead? And then lose energy and then zoom ahead and then lose energy and zoom ahead everytime it hit that thing? [gesturing to show an imaginary Hot Wheels circuit and the zooming effect]

Nobody responds, and Ben decides to move onto the roleplay.

**Analysis**

Following Mona’s question, not too long afterward, Dave questioned the circuit behavior in relation to program execution. The question surfaced Dave’s confusion about stopping or letting electrons flow in juxtaposition with the idea that conductive materials always have free electrons. Dave’s question got Ben to rethink his notional machine that was frequently used during the unit—i.e., circuits as loops of free-flowing electrons and programs as controlling the flow (e.g., 031021, 031621, see Table 5). In response, Ben almost instantly thought of another analogy to explain how electrons get energized in the loop just as Hot Wheels cars get energized in a loop.

In addition, a month later during the unit, Ben revised his notional machine and discussed it with the class (042221). Here, he clarified that “electrons don’t flow in an inner tube. I used that example because it was the only one that I could think of at that time.” He said he had a better explanation and introduced current flow as “a jolt or a push” that electrons experience, just as people would experience if “lined up putting their hand on the back of the person in front.” He asked the class to imagine the current flow as what happens if someone “came from the very back and PUSHED really hard” Dave called it a “chain reaction” while Karla compared it to the “domino effect,” yet again contributing to the construction of notional machine by confirming their
familiarity with the phenomenon. Overall, students revised their conception of the internal behavior of electrons during program execution.

Throughout this activity, learners’ constant meaning-making with notional machines was visible as they adopted notions discussed earlier in the unit (e.g., Mona in episode 1), asked questions (Mona and Dave in episode 3 and 4), and revised their understanding (e.g., Dave in episode 2). Students’ sense-making of program dynamics, being made visible through notional machines, also led to the teacher revising his notional machine during the unit (e.g., episode 4).

Activity 2: Playing out Program Execution

Summary

In this activity, the class collaboratively enacted the notional machine program as a play. Unlike the earlier activity where students verbalized their conceptions and asked questions, learners in this activity took on roles as different parts of the circuit while playing out cycles of program execution iteratively. While they made visible and revised their understanding of program execution through embodied gestures, they observed their peers and the teaching team during the roleplay, and agentically engaged with program dynamics.

TS Transcript Screenshot

1:08:58 Ben: Okay, I am the CPU and you are all going to be my LEDs. I am going to tell you to do things and you are not going to stop until I tell you to stop. You have to continuously do it. And, TA, you are going to be my switch. I need someone to be my switchState. Who wants to be the switchState?

Dave types in “If no one wants to do it I’ll do it”

Ben: Dave, you are the switchState. Awesome. Here’s what you are going to do—You are going to read the TA. If she’s HIGH thumbs up, if she’s low thumbs down. But, you are going to read only when I tell you to and you are not going to update until you read again.
Ben: So, there we go. Int led gets 10. That’s stored in my memory. Int switchy gets 21. Stored in my memory. pinMode led is an OUTPUT. Okay, so, I am activating pin 10. pinMode switchy INPUT. Okay, I am activating that pin.

Ben: Loop! DigitalRead switchy [in a high tone]. Okay, everyone, look at Dave, look at the switchState.

Ben: [TA is continuously changing the state of the switch] If switchState is HIGH, wave. [Dave changed to show LOW, the current state of the switch]

Ben: No no. Dave, you can’t change till I ask you to change. [Dave nods and corrects his gesture to stick to HIGH, the value he had read originally]

Ben: If switchState is HIGH, wave, everybody wave! I need everyone to wave. Don’t stop waving. I will not even go to the else because this was true. So, now, I go back up to the top.
Ben: digitalRead switchy. Excellent. SwitchState is LOW. If switchState is HIGH, this is not true! Keep waving, we are not done waving yet.

Ben: Else if switchState is LOW [by now, the switch was changing but Dave stayed steady as LOW]. This is true, macarena.

Ben: Everyone do the macarena [does the demo himself].

…. [the class does three more rounds with changing switchState values; there is a sense of fluency that’s achieved since students are not requiring any reminders about what to do or to keep their states, nor is Dave requiring any reminders]

Ben: And, we are done! This goes on forever! Mona and Adyan show off their lights responding to the switch. Others are still programming theirs.

Analysis

While interacting with the notional machine *programs as plays*, learners took on agentic roles while interacting with program dynamics during the roleplay, acting either as lights or as the variable storing the state of the switch. Roleplaying program execution instead of directly running the program made visible student conceptions through embodied actions and let the class slow down the dynamic nature of programs and investigate it closely. In particular, Dave volunteered as the switchState variable, a variable whose value is consequential for the outcome of the program. During the roleplay, Dave demonstrated his understanding of how the variable gets
updated and also revised it. At the beginning of the roleplay, although Ben had instructed him to change his value only when asked to do so, Dave, at first, updated the value even when the microcontroller was not executing the digitalRead() statement, surfacing his naive conception of how the value updates in relation to program execution. However, Ben intervened and reminded him to not change and Dave acknowledged. Similarly, students acting as lights also required reminders to make sure they were doing their action until they were asked to change—both referring to how programs affect circuit behavior during execution. However, by the second iteration, the whole class learned their respective roles i.e., how circuits behave in relation to program execution, and demonstrated their fluency by running the roleplay three more times without any reminders or corrections by the teaching team. Right after the roleplay, with very little time left during the class, Mona and Adyan also wrote a similar program with a switch and made two different light patterns for the first time in the unit.

A few minutes later, all students reflected that roleplaying the program execution helped them understand program dynamics better. In Mona’s words, roleplaying “helped [her] understand what the program is doing better... To code the microcontroller, I would have to think like one and look at the code from the perspective of the microcontroller.” Dave reflected, among other things, that he learned “that the computers only read a switch when they are told to, and not continuously as they read the code.” Juan had also “enjoyed” the experience of roleplaying as lights and reflected that “we need digitalRead reads [sic] the value from a specific digital pin, which would be our inputs.” Overall, the students’ reflections in addition to Dave’s revision of his conception highlight how program as a play notional machine allowed learners to interact with program dynamics by making their thinking visible through embodied gestures.

**Activity 3: Reasoning Two-button Program**

**Summary**

A month later (041621), learners continued to adopt notional machines that were discussed earlier in the class and extended them to understand more advanced programs involving multiple
digital inputs. Notional machines here served to broaden the activities that learners were able to engage in (Vygotsky, 1978). In this activity, Ben presented a pre-written program and invited learners to reason selected lines of code. Learners explained their reasoning about the runtime behavior of different parts of the program adopting previous notional machines such as variables as remembering values and program as a play.

Ben paused at line 24, with the digitalRead() statement, to assess students' understanding of program dynamics and their adoption of previously discussed notional machines while reasoning programs. In responding to Ben's inquiry, students including Mona, Karla, and Dave clearly adopted notions from the earlier roleplay of one-switch program and how digital input would be processed.

**TS Transcript**

17:13 Ben: I would like you to go to line 24 of the code [on a shared document with the code]. So, at this point, everything is set up. We have declared everything. Our LEDs are OUTPUTs, our buttons are INPUTs. So, what is happening in lines 24 and 25? Does anybody remember from the last time we did the switch?

[Mona raises her hand]

Ben: Why do we need to have button1State. Mona, go for it.

Mona: So, it’s just basically reading the buttons to see if it’s being pressed or not.

Ben: How come the Circuit Playground does not know automatically that the buttons are pressed or not?

Harem types “you are telling the cpu to read the state of the buttons.”

Ben: Yes, why do I need a button1State though? Why do I have or why do I need a variable for this?

Dave types “Because the computer does not store memory well... Something like that”

Ben: [laughs at his response] You are actually pretty close... Okay, let me do something, I am going to delete. Everyone see on line 24, I just deleted the int button1State. Now, what would happen if I ask the computer, right now: Hey, read button1 and whatever comes out if that thing is HIGH... is the computer going to know what I am talking about?
Harem types “it won’t know what state it is if we don’t tell it”
Ben: Why not? I read it. The computer read button1. Why
won’t it know what it was?
Mona types “it didn’t store it anywhere”
21:14 Ben: [reads out Mona’s response] We don’t have a way to
be able to talk about it again.
Dave types “Not without the button state because it doesn’t
know what to do with the command”
Ben: So, we name things so that we can refer to them again
later.

Analysis
Even a month later, students adopted notions from earlier discussions about processing digital
input signals and reasoned a program with two buttons. Mona, Dave, and Harem took turns to
respond to different questions posed by Ben about the two lines of code that read and stored the
digital inputs in the program. Although this was their first time programming with multiple digital
inputs, learners engaged with program dynamics while thinking of underlying behaviors in terms
of simplified ideas discussed earlier during the unit. For instance, Mona thought of the lines as
“reading the buttons” just as Harem thought that the lines were “telling the cpu to read the state of
the buttons,” when asked to reason the two digitalWrite() lines of code. Further, specifically when
discussing the variables that are saving the digital input values, Dave, although not sure,
suggested that the variable is helping the computer store the value in its “memory.” Harem and
Mona built on this idea and reasoned the buggy scenario Ben created. When the variable was
deleted, Harem hypothesized that we would not know the state of the button and Mona described
the reason as not storing the state anywhere. Student reasoning of program lines connected back
to Ben’s notional machine of variables as remembering or storing value, communicated multiple
times during the unit. Overall, this simplified notion of variables as “remembering” values allowed
learners to engage with program dynamics by reasoning the different parts of the program and
their behavior during execution.
Activity 4: Predicting Circuit Status Based on Program Execution

Summary

In this activity (041621), students were asked a challenging question, predicting the state of the circuit components at a particular execution time. Learners here not only made predictions, but also justified them by adopting their understanding from previous notional machines. Karla even highlighted how she specifically drew from her experience of roleplaying earlier during the unit to answer the question.

29:00 Ben: I have a tricky question for you. If button1State is currently HIGH when we are in line 28, does that mean the button1 is still pressed? [Ben types up the question in the chat window as well]
Adyan says no, Mona types “no, it was pressed in the past,” a few other students including Dave type no.
Harem has typed: “no because it won’t know to recheck until the loop is done”

Ben: Okay, we see a lot of NOs here. I want to know why.
Harem, so you’re saying that we only know what it is when it got checked?
Mona types in “it was pressed in the past, and the state was stored in the variable and the variable doesn’t change”
Karla said “we talked about this last time with the flashlight (I think)”

Analysis

The class had discussed programming with a single digital input. Roleplaying the program as a play notional machine had allowed students to connect program execution to the state of the circuit components during runtime (031821). However, in the activity a month later, Ben asked a prediction question that required students to make similar connections and hypothesize the state of the circuit given the value of a particular variable during execution. Ben called it a “tricky question” since the class had to predict the state of an input device at a particular time during program execution, unlike the usual questions of predicting the output device states such as lights. To Ben’s surprise, learners not only responded accurately but also articulated their
reasoning. In particular, Mona and Harem stated that the button state cannot be derived from the value in the program, connecting the temporal relationship between hardware and software during program execution within e-textiles. Karla further connected her reasoning to the activity with flashlights—i.e., roleplaying program execution of a one-switch program discussed above (031821). Reasoning in this context required students to go beyond the light pattern outcomes that were usually discussed and instead reason the relationship between digital inputs and the variable in the program.

Overall, learners not just received the notional machines but actively engaged with them while making sense of program dynamics. They took on agentic roles, making their understanding about program execution visible, revising them, adopting prior notions, and even asking clarifying questions while making sense of program dynamics through notional machines. The collection of notional machines adopted during this unit (see Table 5) had become shared functional concepts for the class community and had transformed from a concept of program dynamics presented by the teacher to the concept for problem-solving among learners (Bowers et al., 1999). Not just revising their own notions, learners also led the teacher to revise his notional machines to more accurately represent underlying behaviors.

Limitations

Despite promising findings, the study was limited in a few different ways. First, this study was conducted in an online setting due to the ongoing COVID-19 pandemic. The class met on Zoom, which offered very limited ways of capturing student presence. Only students who turned on their video, audio, or engaged in the chat were visible during the class. With the ongoing pandemic, the teacher and the school desisted from mandating any kind of participation (e.g., compulsory video on). This challenged data collection of students who did not or could not participate actively on Zoom. Unlike physical classrooms with videos that captured student presence anyway, data collection within online settings resulted in very few students leaving consistently rich data traces.
Second, students were physically distanced, taking classes from their individual homes during the pandemic that made collaborations and interactions among peers difficult. Although the teacher encouraged students to establish informal ways of staying in touch with each other by exchanging phone numbers and social media handles, student interactions through any of these were not visible to the teacher or me. In order to mitigate these to an extent, the teacher adopted Seesaw, an online platform for students to share projects in progress and engage with each other by leaving comments and likes. However, very few students utilized this option to engage with one another; rather, they treated the platform as a portal to “submit” their project progress like “class assignments.”

Discussion and Conclusion

The findings presented above highlight students' agentic roles while making meaning of program execution within computing classrooms. Students actively engaged with the teacher's notional machines as they adopted them, questioned them, and revised their understanding of program dynamics. Findings from this analysis make a three-fold contribution: they highlight the mediating role of notional machines in supporting agentic learner interaction with program dynamics within introductory computing classrooms, their role in sustaining joint meaning-making within computing classrooms, and their evolving nature as a result of continuous interactions between the learners and the teacher. Unlike prior studies that had only captured learner conceptions of program dynamics through think-aloud interviews and quizzes, this study provided one of the first accounts of the student learning process with notional machines within computing classrooms analyzed through a sociocultural lens (Sorva, 2020; Guzdial et al., 2019). Furthermore, this was one of the first attempts to capture student interactions with notional machines within a physical computing context where programs are accompanied by physical circuits (Krishnamurthi & Fisler, 2019).
The Mediating role of Notional Machines

Though researchers had hypothesized the mediating role of notional machines within computing classrooms (Fisler and Cunningham in Guzdial et al., 2019), the qualitative analysis of classroom accounts presented in this paper is one of the first attempts to capture and analyze it in practice. Adopting sociocultural theory to analyze learner interactions with notional machines over an extended period highlighted the mediating role of notional machines. Notional machines, though simplified ideas of program execution, provided functional tools for novice learners to engage with program dynamics while being socialized into disciplinary practices such as reasoning, writing, and debugging programs within the context of e-textiles. As seen in the accounts above, learners anchored their meaning-making of program dynamics through the notional machines such as variables as remembering values, programs as plays, and circuits as electron loops. These simplified approximations made abstract, invisible concepts accessible to learners. Further, they allowed learners to participate during the unit in agentic ways despite being beginner programmers. However, future research needs to study notional machines in other classroom contexts to capture further nuances such as the role of student and teacher expertise in shaping the mediational role of notional machines.

Notional Machines Sustaining Learner Meaning-making

Unlike earlier studies that had captured "atomic" or individual instances of notional machines and individual students' use of them in independent problem-solving activities (Fincher et al., 2020), this study captured teaching and learning over an extended time and considered classroom activities of reasoning, writing, and debugging programs as the unit of analysis. By providing shared language and conceptual tools, notional machines allowed learners to make meaning of program dynamics as a community of learners throughout the unit. As seen in the accounts above, notional machines served as a functional concept for the class to engage with program dynamics during the unit. Over time, keywords like "remembering" or "storing" related to specific
notional machines gathered significance in this high school class. They signaled particular meanings for learners, thereby sustaining joint meaning-making of programs throughout the unit. Advancing concepts within curricular units required learners to constantly engage with program dynamics and extend their conceptions to reason more sophisticated outcomes, in this case involving multiple input and output devices. Learners were noted adopting notional machines from prior discussions and learning activities (such as roleplays) to reason about newer programs and predict their outcomes. Notional machines situated within classroom communities not only supported learners' engagement in individual activities but also sustained their participation across the unit by providing specific ways of making sense of program dynamics within the context. While this study was situated within an introductory high school unit, future research should explore this function of notional machines within undergraduate and graduate classrooms over extended durations where notional machines have already been predominantly identified (Fincher et al., 2020).

**Evolution of Notional Machines**

Prior investigations of notional machines as conceptual models provided by teachers had shed very little light on how they shift and change over time, affirming a static nature to notional machines (as seen in Fincher et al., 2020 mode, Fig. 12). But, examining learning with notional machines for extended periods in this analysis captured the dynamic nature of notional machines. It widened the scope of examining notional machines, not just as static computing models but as closely tied to the classroom interactions, bearing specific meanings and significance for the community of learners within classrooms. For instance, it was visible in how the "circuit as electron loops" notional machine shifted in interaction with students in the accounts above. Though the teacher may start by presenting a particular approximation of program dynamics, observing students' engagement with these simplified ideas can lead to changes in these notions to befit its purpose of supporting students. Therefore, future research should capture not only
individual instances of notional machines but also their potential evolution over time and the role of teacher-student interactions in shaping it.

This study provides one of the first classroom accounts of high school students learning with notional machines throughout an introductory computing unit. Though analogical and metaphorical explanations have been historically studied within science and mathematics classrooms (e.g., Lakoff & Johnson, 2008), notional machines support student learning in similar ways within computing classrooms. Despite being simplified approximations of program dynamics, they allow novice learners to engage meaningfully within the computing discipline. Further, unlike studies that have projected notional machines to be fixed models to represent a particular part of program dynamics, the accounts above highlight the situated nature of these machines, shaped by the programming environment and tailored to the community of learners and their interactions with them.
“IT IS PROGRAMMED:” SHIFTS IN HIGH SCHOOL STUDENTS’ CONCEPTIONS OF SENSOR-BASED DEVICES AND TOYS

An earlier version of this chapter is published.


Chapter Summary

Advancing conceptions of computing system internals—including its structural composition and dynamic behavior—is key to learning computing and critically engaging with it. However, novices' informal ideas challenge their engagement. In this chapter, I qualitatively analyzed ten students' think-aloud interviews before and after a 14-week-long online electronic textiles unit. In these interviews, both the times, I asked students to describe the inner workings of a pair of functionally similar computing artifacts—an everyday sensor-based device such as automatic soap dispenser and an interactive toy—to understand shifts in their understanding of the structure and behavior of computing systems. The analysis revealed that, post-unit, the students had developed advanced conceptions of the behavior and structure of computing systems. They frequently accounted for computing and explained the internal dynamics using data and control flow. We discuss the implications of our findings for future research to support novices within introductory computing contexts and to promote critical computing learning.

Introduction

A primary objective of introductory computing education is to help learners develop a conception of layers of abstractions between software and hardware that make everyday computing systems functional (CSTA, 2017). With designs of devices such as automatic soap dispensers, spirometers, and facial recognition software critiqued for biases against marginalized peoples
(e.g. Benjamin, 2019; Costanza-Chock, 2020), understanding the internal dynamics of computing systems is important to recognize how social aspects, such as race and gender, shape programs and their outcomes. Though these everyday devices are recommended as conversation starters for critical computing pedagogies (Costanza-Chock, 2020), we know very little about how novices think about computing devices, their designs, and their internal workings. Understanding the internals of computing systems, such as how control and data flow within them, is quintessential to reason program execution while reading, remixing, writing, and debugging programs (duBoulay et al., 1999).

However, the invisibility of internal connections and dynamic flows in computing systems challenges learners. Studies have shown that novices think of devices, like digital thermometers, in simplistic ways. They directly connect visible inputs and outputs such as buttons and displays, respectively, without understanding any details about the inner connections or mechanisms (e.g., Pancratz & Diethelm, 2020; Cederqvist, 2020). Students’ conceptions of engineered artifacts, such as sensor-based soap dispensers, involve three parts: structural composition and interconnections, functionality of these parts and the whole system, and the behavioral dynamic, i.e., the inner workings such as control and data flows that cause observed outcomes (Bhatta & Goel, 1997). As much as uncovering internal workings are essential for learning computing, novices across age groups start with naïve conceptions based on their experiences as users. From inaccurately attributing computers with human-like reasoning abilities (Pea, 1986) to conceiving physical computing devices in simplistic ways (Jayathirtha & Kafai, 2021a), novices’ informal conceptions of computing systems often challenge their learning. Further, these informal conceptions limit novices’ understanding of the role of data and programs in shaping observed outcomes, thus restricting their engagement.

Tool designers have tried to support novices’ computing learning by providing learners with concrete machines to program. Physical computing construction kits include microcontrollers, inputs, and outputs for learners to make tangible artifacts and program them (Blikstein, 2013). Construction kits such as electronic textiles (hereafter e-textiles), adopted within
constructionist introductory curricular units, introduce learners to computing by providing them opportunities to design and handcraft circuits and program them (Kafai et al., 2019). These tools afford opportunities for learners to make artifacts that are functionally similar to sensor-based everyday technologies (Przybylla & Romeike, 2014). They make structural aspects of the computing system transparent by revealing circuit elements and their connections, although they hide program execution and its interaction with circuits (Buechley et al., 2013). While physical computing is gaining popularity within introductory high school classrooms (e.g., Kafai et al., 2019), studies that examine high school students’ conceptions of these systems are only emerging (e.g., Pancratz & Diethelm, 2020).

In this chapter, I conducted and analyzed student think-aloud interviews (Ericsson & Simon, 1998) before and after an introductory e-textiles unit within a high school online class. I asked: How did student conceptions, particularly the structural and behavioral understanding of physical computing systems, shift at the end of the unit? I chose the interview artifacts—sensor-based everyday devices and interactive toys—to be functionally similar and aesthetically different to allow for analytical comparisons of student responses. Unlike earlier studies that reported student conceptions at a single time point (e.g. Cederqvist, 2020; Jayathirtha & Kafai, 2021a), this analysis of student conceptions at two time points sheds light on the shifts in novices’ understanding. It also informs curricular design efforts for deepening computing and critical pedagogies.

Background

How Students Conceive Physical Computing Systems

Student conceptions of physical computing artifacts, such as automatic soap dispensers, pulsometers, and student-made e-textile projects, consist of structure, behavior, and function (Bhatta & Goel, 1997; Jayathirtha & Kafai, 2021b). The physical parts of the system and their interconnections make the structural composition; roles of different elements and the overall...
purpose of the system cover the functional aspects; and the underlying logic resulting in outcomes through control and data flows accounts for the behavior (Bhatta & Goel, 1997). In the case of sensor-based devices and interactive toys: (1) the structural aspects include components such as microcontrollers, sensors, lights, and their physical interconnections; (2) the functional aspects consist of the roles of each of these components as inputs and outputs and the overall purpose of the artifact; and (3) the behavioral aspects include the underlying program logic that processes inputs and controls outputs to cause observed outcomes.

There has been a historical interest to understand learners’ conceptions of computing devices around us, including studies interviewing students about washing machines, coffee makers, stoplights, and smartphones, to name a few (e.g., Przybylla & Romeike, 2014; Cederqvist, 2020). Across these studies, novices’ limited conception about these systems stands out. For the most part, the visible input and output components are the most accessible to students (Simplistic in Fig. 14, left). For instance, Pancratz and Diethelm (2020) observed 200+ German middle and high schoolers demonstrating a range of naive conceptions about the inner structural composition of physical computing systems, either missing connections or drawing simplistic connections between visible aspects. Further, a few other studies (Cederqvist, 2020; Jayathirtha & Kafai; 2021a) noted how students within introductory computing programs had a limited understanding of computation within computing devices. They either included a computing component, or just acknowledged the presence of programs without clear connections with the rest of the system or how programs may coordinate with different elements to cause the outcome (Limited in Fig. 14, left). Across these studies, a more advanced conception involving all the parts, their interconnections, and a clear role of computation was rare among middle and high school students (Advanced in Fig. 14, left). Concepts of control and data flow—how inputs such as touch and light intensity get transformed into numerical values for further computation to cause different outcomes—were almost inaccessible to novices (e.g., Cederqvist, 2020). Such conceptions indicate advanced understanding across layers of abstractions between hardware
114 and software, and the learners’ preparedness to engage with the internal design of artifacts and programs.

Figure 14: Representation of categories of student conceptions (left) and a sample e-textiles student project showing code on the computer and a connected physical artifact (right).

What Students Learn with Physical Computing Kits

Physical computing construction kits have historically played a central role within introductory K-12 computing education. From Papert’s turtle (1980) to robots to the recent microcontroller-based kits, introducing young learners to computing has involved a physical artifact in addition to on-screen computer programs (Blikstein, 2013). Tools such as e-textiles have allowed learners to design personally-relevant artifacts and program them, bringing sewing and crafting into electronics and computing (Fig. 12, right). Handcrafting circuitry further makes the structural
aspects of these systems transparent. E-textile artifacts, such as touch-sensitive murals and temperature-sensitive lunch bags, come very close to several sensor-based computing devices around us, consisting of inputs, outputs, and processing units. Kits like these allow learners opportunities to draw connections between their projects and everyday computing devices (Przybylla & Romeike, 2014).

Learning with physical computing systems requires one to go beyond the visible components and develop conceptions about internal structural and behavioral details to understand program execution, specifically the different interconnections between parts and data and control flow while programs run (Ananthanarayan & Boll, 2020). For example, learning with e-textiles requires one to identify structural components such as microcontrollers, and input/output devices. Further, they need to understand interactions between circuits and programs such as analog sensors producing values, programs using variables to store and compute with these values, and microcontrollers executing programmed logic to cause different outcomes such as sound and light patterns (DesPortes & DiSalvo, 2019; Jayathirtha et al., 2018). However, these invisible abstract ideas challenge learners. Jayathirtha and colleagues (2018), similar to DesPortes and DiSalvo’s (2019) observations of novice undergraduates, noted high school students’ difficulties conceptualizing different behavioral dynamics between textual programs and circuitry even in simple physical computing projects. Learners reported problems understanding how internal processes, like button presses, get processed to produce light patterns in e-textile projects, despite having a tangible circuit to interact with. In summary, though physical computing kits can potentially support student learning of a wide variety of sensor-based devices, there is a need for further exploration of how learners make connections between different devices and how their conceptions about internal structures and behaviors change.
Methodology

Context and Participants

The study was conducted at a public charter high school located in a U.S. west coast city. An experienced high school computing teacher was teaching an online e-textiles unit within the Exploring Computer Science (ECS) course. It was a constructionist physical computing unit where learners make four tangible projects and gradually explore circuit and computing concepts (Kafai et al., 2019). To highlight the internal dynamics of physical computing systems, the teacher infused a variety of additional explanations and learning activities throughout the unit. He used metaphors such as Circuit as Electron Loops to describe circuit behavior and plays to concretize abstract circuit- and computing-related ideas like electricity and information flow during program execution. In this chapter, I focus on analyzing shifts in student conceptions about physical computing systems before and after the unit.

For interviews, the teacher chose ten out (5 male, 5 female; all except one identified as non-White) of the 24 consenting students (from a class of 34 students) with different degrees of classroom participation (3 active, 4 intermittent, 3 reserved). The teacher identified the students based on his experience teaching this class for a semester. All ten students were formally learning to program for the first time and had around ten weeks of experience with Scratch programming as a part of ECS. While two of them had additional experience with Python and C++ due to prior individual exploration, all the ten were new to physical computing.

Data Collection and Analysis

Think-aloud protocols (Ericsson & Simon, 1998) were adopted for pre- and post-unit online interviews to capture students’ structural and behavioral understanding. Each interview (lasting 20-30 mins each) had two sets of questions: the first about the working of a sensor-based everyday device (automatic soap dispenser pre-unit and pulsometer post-unit) followed by another set interactive toy (see Fig. 15, left). All four artifacts were functionally similar, i.e., they all
involved analog sensors and digital outputs but differed aesthetically (see Fig. 15, middle). The everyday household devices talked about in the interviews were used during the COVID pandemic but not discussed in class. The toys, on the other hand, were similar to student e-textile projects but more sophisticated. Students were given the functional description of the artifacts during the interviews and asked questions such as: “What do you think is inside? How do you think it works?”. They were encouraged to draw and talk out loud (Ericsson & Simon, 1998).

Student responses across pre- and post-unit interviews were thematically analyzed for structural and behavioral details (see Fig. 15, right). Verbal responses and drawings were annotated with on-screen gestures. Based on earlier work (Fig. 14, left), I generated a codebook to identify structural and behavioral details and qualitative distinctions (*simplified, limited, advanced* as seen in Fig. 14) with examples from student interview transcripts. Students' responses about every artifact were parsed for structural and behavioral explanations, and every description was coded to capture qualitative differences (*simplified, limited, advanced*). For instance, structural and behavioral aspects were separated in Dave’s response (see the light and dark shades in Fig. 14, right), and each was further classified as *simplistic* since there was no mention of any computation mediating inputs and outputs in the structural or behavioral explanations. Structural explanations that included a computing component but lacked apparent interconnections between parts were categorized as *limited*. Explanations with clear interconnections between computing, input, and output devices were coded as *advanced*. Within behavioral explanations, the mere mention of computing without further details about its role was coded as *limited*, and explanations with any further elaboration about how computing mediates the system behavior were categorized as *advanced*. Along with a graduate student, I discussed the codebook and analyzed two students’ (20% of the dataset) pre- and post-unit interviews. They agreed upon 15 out of 16 (93.75%) codes and resolved the discrepancy through discussion. They analyzed the rest of the interviews independently, verified their responses (~90% agreement), and clarified discrepancies.
I further inductively analyzed behavioral explanations to capture qualitative differences, specifically for how students considered or not the control and information flow in these systems. Four key themes emerged: (a) behavioral descriptions which involved only circuit-based mechanisms; (b) explanations that treated programs and circuits as disconnected; (c) explanations that only accounted for control flow (such as basic conditional logic) within the systems; and, (d) descriptions that include both control and data/information flow between programs and circuits during program execution. The first two closely overlapped with simplistic and limited explanations. The latter two themes were evident within the advanced category, i.e., accounted for computing by highlighting different aspects of its dynamic nature. For instance, Dave’s behavioral explanation pre-unit belonged to the first theme involving only a circuit-related mechanism: a “contraption” activated by a sensor directly to “dispense some soap,” thinking of the sensor like a simple electric switch. A few responses that articulated the role of computing in terms of conditional logic, thinking of computing as organizing the control flow, without any mention of data, belonged to the third category. Explanations that elaborated in terms of data, i.e., either how inputs get transformed to data or how values get computed or compared to cause outcomes, were grouped under the final theme. The same graduate student who partnered during analysis was presented with the themes and example excerpts from student responses, and they
agreed with most of them (85% agreement). Disagreements were resolved through discussion, and codes were updated to reflect these changes.

Findings

Overall, students’ structural and behavioral understanding of sensor-based devices and toys shifted from simplistic or limited pre-unit to advanced post-unit (see Fig. 16, central table). Structural explanations were mainly limited to input and output devices pre-unit. Behavioral explanations either involved only electronic interactions between circuit elements or accounted for control flow similar to conditional logic in Scratch. However, post-unit, structural explanations included computational components and clearer interconnections. Further, behavioral explanations included data flow details involving sensor values and their processing.

Figure 16: A visualization of distribution of student responses across structural and behavioral details (table) and themes and frequencies within behavioral explanations pre- and post-unit (graphs either sides).

Towards Advanced Structural Understanding

Most structural explanations pre-unit missed computing, more so in the case of automatic soap dispensers (Fig. 16, Simplistic column along Structure row). Although the toy’s structural composition was more accessible pre-unit, most responses still lacked interconnections between different components (14 out of 20). However, post-unit, students’ structural explanations across artifacts increasingly involved a computational component in their composition and connections
(table in Fig. 16, Structure row). With a few exceptions, most descriptions included computing-related components such as “motherboard,” “circuit playground,” and microcontrollers, and drew elaborate interconnections between input, output, and processing units.

**Bringing Computation into Structural Composition**

Across both interview artifacts, explanations about structural composition significantly got more sophisticated post-unit. Pre-unit, more than half of the explanations (14 out of 20) missed any computing component or included one without a clear role. However, at the end of the unit, most descriptions (15 out of 20) involved a separate microcontroller connected to input and output devices. For instance, pre-unit, Dave described the internals of the soap dispenser as involving only a sensor and a mechanical "contraption" for letting the soap out (Fig. 16, a.; 020521, pre-unit). Even when students mentioned a programmed component or a “board,” it did not involve definite connections between the different parts (4 out of 20). For instance, Sasha tried to account for a programmed “code playground” with a “motion detection” [drawing the idea from the materials she had received ahead of the unit] in her structural description of the soap dispenser but barely drew any connections with the rest of the system (020521, pre-unit). Although there were a few descriptions that included a programmed component connected to inputs and outputs (advanced descriptions in pre-unit in Fig. 16), like how Mona drew out a “board” connected to lights and a “music box,” they were more so in the case of the toy than the soap dispenser.

A majority of students’ explanations post-unit were advanced in nature. As evident in Dave’s post-unit explanation, the machinery inside the pulsometer that allows them to sense and display pulse rate was far more sophisticated (Fig. 17, b) than his simplistic structural description for a similar sensor-based device, the automatic soap dispenser, pre-unit. For the pulsometer, he articulated the presence of a “motherboard” [blue circle in his drawing], a “sensor” that can sense the pulse [red circle in his drawing], and the screen where the pulse will be displayed [black outline]. A few explanations (4 out of 20) were still limited because they included a computing component without any mention of clear interconnections. Furthermore, the only exception was
one of the explanations that were simplistic post-unit (Karla’s pulsometer only had a sensor). Overall, students’ structural explanations moved towards being more advanced at the end of the unit.

**Figure 17:** (a) Dave’s annotated drawing of internal structure of an automatic soap dispenser pre-unit; (b) Dave’s pulsometer internals post-unit annotated; (c) Karla’s structural drawing of the bird toy pre-unit (d) Karla’s drawing of fox toy internals post-unit.

---

**Towards Elaborate Structural Interconnections**

Besides structural composition, there was a significant difference in how students explained interconnections post-unit compared to pre-unit. Unlike mentioning different parts without specific connections between them in the pre-unit interviews, students made more elaborate connections post-unit. For instance, in his description of how a pulsometer works, Dave clarified that the sensor and the screen will be connected to the “motherboard” to receive values and display them onto the screen. Such an explanation is similar to Karla’s responses (Fig. 17, d), where she drew out all the parts and interconnections post-unit. Adopting circuit-drawing practices from the class, students drew out separate positive and negative connections between the lights and the microcontroller, sometimes even color-coding them (see the red and blue lines in Dave’s and Karla’s drawings in Fig. 17). These explanations are unlike Karla’s pre-unit response that involved a “disk with code” connected to the “pressure sensors” at the hands of the toy bird, without any other actuators or connections with the “disk” (Fig. 17, c).

Students drew most of these interconnections based on their functional understanding of these different parts, as evident in their verbal descriptions that accompanied their drawings. For
instance, students noted connections between the microcontroller and a display screen or actuators like lights to “send signals” to produce outcomes like printing out the pulse rate in the pulsometer or making the lights blink in the toy. At the same time, connections between sensors and the microcontroller were established so that signals could be sent from the input devices to the controller for further processing. After identifying the type of sensors (mostly light or sound sensor in the case of the pulsometer, and pressure/touch sensor in the case of the interactive toy), students connected them to the processing unit so that they could send signals for further information processing. Students made similar connections while constructing their projects in the class, sewing connections between the microcontroller and actuators like LED lights using conductive thread. Overall, their structural composition and interconnections saw a significant shift across the artifacts at the end of the unit, although the toy was more accessible initially.

Towards Advanced Behavioral Understanding

Unlike structural understanding, students’ conception of both systems' internal behaviors was simplistic or limited (Behavior row in Fig. 16) pre-unit. However, post-unit, this shifted to more advanced understanding within behavioral descriptions across everyday technologies and e-textile toys, more pronounced in the case of the latter. Further, students included more data and control flow details in their internal working explanations post-unit.

Towards Advanced Behavioral Explanations

Before the unit, most behavioral explanations (15 out of 20) were either simplistic or involved computing in a limited fashion. For instance, in the pre-unit interviews, these students described the inner workings in terms of simple electronic or mechanical setups. Santos explained the underlying mechanism for the toy as “two pieces of wires, when squeezed, touch and send a signal to the light. They turn brighter when you squeeze them harder” (022421, pre-unit). Juan, just like Dave above, explained the working of automatic soap dispensers as a simple mechanism of “the motor spinning around, probably spinning other components that enables the lid to open”
A few explanations (6 out of 20) articulated computing as mediating interactions between inputs and outputs, although without details. For example, pre-unit, Sasha identified that there would be a program controlling the toy and could not explain its role further in causing the light patterns or music.

However, post-unit, students’ explanations involved more computational details, moving between the different layers of abstractions within the hardware and the software. Fifteen of the 19 descriptions had computing mediating data flow between the input sensors and the actuators such as LEDs or the display; only 5 of the 19 explanations were simplistic or qualified. For instance, post-unit, Dave explained how data gets processed with the help of a “motherboard,” imagining operations such as counting and waiting for a specific time to calculate the beats per minute in the pulsometer. He said, “[the device] collects the data... sends that data to the motherboard... so, when [the motherboard] receives a vibration range in a certain amount of time... if those values are in quick succession, it would be assigned a number, and it would display the BPM” (052421, post-unit). This was in contrast to his explanation of a similar sensor-based soap dispenser pre-unit, where he had described the simplistic inner working with a mechanical contraption and a “sensor” that would work similarly to a switch. Overall, post-unit, learners acknowledged the role of programs in controlling the behavior of the system more often than they did pre-unit.

More Data Flow Explanations

Students' behavioral explanations, post-unit, moved from predominantly circuit-based or control-flow-based to more data-flow-based (see Fig. 14, graphs on either side). In their initial behavioral explanations, a majority either involved only circuit-related aspects (7 out of 20) or treated circuit and program as separate entities (3 out of 20) or, at best, thought of it in terms of conditional control flow, drawing from if-blocks they had just learned in Scratch (9 out of 20). Students borrowed Scratch constructs, such as if-blocks, and constructed explanations substituting Scratch sprites with input buttons, sensors, and actuators, without discussing details about circuits and
programs interacting. For example, Sasha’s pre-unit explanation for how the toy bird would work. She said, "In Scratch, we have like if This sprite is clicked, this will happen… It’s kinda similar to that… like when this button is clicked so then this function happens… quite similar to Scratch in the sense that like when you hit something, then it changes” (020121, pre-unit). Some identified the presence of a programmed component within the toy that dictates the control flow to perform different functions based on human interaction, but without discussing any interaction between programs and circuit elements during execution. For example, Malia explained the toy bird pre-unit as "it would know when you press it softly when it would get the idea that.. it would play different music and do blinking lights. It would just know. There would be a program" (020921, pre-unit). She struggled to connect the program to the underlying behavior, although she identified the existence of a program. For many others (7 out of 20), internal behavior involved interaction between circuit elements only. For instance, when discussing the toy bird, Raul mentioned that "when you press down the button, you know, it’ll move this thing down [draws a button connected to a surface]. And so with just right enough force, it will go to this, but if you press too hard, huh, it might go under or something…" (020921, pre-unit). Overall, based on their limited understanding pre-unit, students’ behavioral explanations were shaped by their on-screen Scratch programming experience in the best case, missing out on details about how programs interact with the circuit components to cause the observed outcomes.

However, at the end of the unit, students moved towards better articulating the mediating role of computing within these artifacts, at least in one of their explanations. A majority of explanations (9 out of 19) included some form of computation with numerical values produced by the sensors as mediating the system’s overall behavior. They either thought of programs as comparing values to make decisions for outcomes or perform arithmetic operations on them, such as counting or incrementing, based on the pulses sensed, just like Dave above. A small group of students, who had earlier thought of programs and circuits as disconnected or had any such relation between them, could think in terms of control-flow mechanism, thinking of programs to remember or relay information from the sensors. Raul, for instance, thought of a pulsometer as
"when it senses that it'll [the pressure or temperature sensor]... It's like, relay to the screen or whatever, or the lights to display this, like if, it was pulse was ninety-five, it'll tell like, yeah, it's in ninety-five. So, display that." (052721, post-unit). However, a small proportion of students (2 out of 10) struggled to explain the behavior of the pulsometer, although they moved up the ladder with the same for interactive toys.

Discussion

Students moving towards constructing advanced explanations of the internal dynamics of everyday devices and interactive toys at the end of the unit is promising in a few ways. It strengthens earlier hypotheses that working with physical computing construction kits helps students connect between their curricular projects and functionally similar real-world technologies (Przybylla & Romeike, 2014). Unlike prior studies that reported school students across a wide range of ages having difficulty unpacking everyday devices (e.g., Pancratz and Diethelm, 2020), our results in this study noted that those differences were less noticeable in high school students. This study was one of the first ones exploring changes in learner conceptions across a semester-long unit. The findings point in a few different directions for future research. They suggest that more work is needed to examine the role of curricular projects, the curriculum, and the context of e-textiles in deepening student conceptions about the inner workings of physical computing systems. For instance, one potential future study can examine how teachers’ explanations of underlying mechanisms in the form of metaphors and other pedagogical tools support learners to understand abstract behaviors. Further, the role of constructionist curricular projects that afforded several opportunities to construct circuits in shaping students’ conceptions of computing systems needs further exploration. At the same time, understanding how tangibility and materiality of e-textiles construction kits support student learning is another aspect to examine.

With limited pre- and post-unit interviews, this study calls for further research exploring student conception development within introductory computing settings. While we captured pre- and post-unit student responses, tracing student thinking and classroom participation during the
unit can shed light on how novices develop, revise, and refine their conceptions as they interact with programming environments and curricular designs. The field can better understand how student conceptions evolve and how they can be supported through intentional pedagogical designs. While physical computing context allows specific opportunities for learners to conceive of computing systems by making inputs and outputs visible, similar explorations within on-screen programming environments will further help us understand such development across computational contexts.

The observed shifts in student conceptions were not a surprise, given they had participated in a constructionist curricular unit for a significant time. And yet, analyzing students’ structural and behavioral conceptions illuminated specific aspects of this conceptual shift. Learners not only developed a more detailed understanding of the structural composition and connections, but they had also developed a language to talk about different layers of abstractions between the hardware and software. Such a trend can be considered a step towards understanding the underlying designs of everyday devices that embody social injustices (e.g., Costanza-Chock, 2020). Thinking about the internal dynamics of devices such as automatic soap dispensers or digital pulsometers in terms of sensor values and their processing makes room for critical questions such as whose data is included and excluded in the device design and what are the implications for different user groups. Such questions can lead to recognizing how social aspects such as race, gender, sexual identity, etc., intersect with the design of these devices and developing computing literacy required to advocate for actionable design changes within everyday computing systems (Costanza-Chock, 2020).
CHAPTER 6
DISCUSSIONS AND DIRECTIONS

Findings from the different analyses highlight how notional machines glass-boxed computing within the introductory computing electronic textiles unit. Investigations to understand teaching with notional machines showed the different notional machines the participating teacher adopted and the forms they took in practice. The notional machines were spread across five themes to simplify both circuit and program behavior for novice learners within the physical computing unit. Notional machines also differed in their granularity of focus and their form or representation. While some notional machines simplified behavior of individual symbols or statements during runtime, others were designed for students to make sense of specific blocks or the entire program execution. Across the notional machines enacted within the e-textile unit, two key forms were noted: verbal explanations and participatory roleplays. Analyzing notional machines in practice showed the multimodal nature of these forms: they were composed of teacher’s embodied gestures, modulated voice, and on-screen representations.

Investigations of student interactions with notional machines during the unit illuminated the mediating role that notional machines played in student sense-making of program dynamics with electronic textiles. Qualitative analysis of student participation during the unit showed their agentic role, such as adopting notions, making their notions visible and revising them, asking clarifying questions, and roleplaying program execution. Further, notional machines sustained conversations around program dynamics in addition to going through revisions in interaction with learners during the unit. Analysis of student pre-unit and post-unit interviews revealed shifts in conceptual understanding about computing systems. Students progressed from having a limited conception of physical computing systems to developing advanced conceptions about the structural and behavioral aspects of physical computing systems. They not only explained artifacts closely related to their own e-textile curricular projects but also everyday sensor-based devices. In sum, findings from this thesis demonstrate the role of notional machines within K-12
computing classrooms in mediating interactions as students make sense of invisible program dynamics, and broader shifts in student conceptions about physical computing devices. They also illuminate potential opportunities for critical engagement with everyday computing devices. Below, I first address limitations of the study before discussing them. In the following sections, I summarize findings from each line of inquiry, and then discuss them in relation to existing knowledge within computing education research and suggest future research directions.

Teaching with Notional Machines

The following questions guided the inquiry about teaching with notional machines: (1) What notional machines did the teacher adopt? (2) What were their forms? Teaching computing with physical computing required the participant teacher to support learners to develop simple and useful conceptions across different layers of the programming environment—circuits, programs, and the interaction between them during runtime. Qualitative inductive and deductive video analysis of seven class periods highlighted the five key themes along which the participant teacher presented simplified notions of underlying dynamics.

**Circuits as electron loops** were adopted at the beginning of the unit when students were making simple electronic projects with either one or a few lights in parallel. This notional machine aided Ben, the teacher, to help students understand how the lights turn on in e-textile projects, and the relationship between programs and different parts of the circuit, such as the conductive thread, battery, and switch. Further along the unit, other notional machines like *Microcontroller pins as programmable gates*, *Programs as plays*, and *anthropomorphized microcontroller* were introduced when making connections between programs and circuits during program execution.

*Microcontroller pins as programmable gates* particularly linked circuits with programs, describing circuit behavior in relation to the execution of specific program statements (e.g., `digitalWrite()`, `digitalRead()`, `delay()`, etc.). *Programs as plays* helped the class think of program execution as a play with definite characters, roles, and responsibilities, with scripts enacted during the play. Using role play, the teacher demonstrated how the microcontroller controlled the
lights and updated variables while reading switches, buttons, and sensors. Such a notion provided the class a way to connect the entire program execution to the different outcomes observed in the circuit.

Furthermore, anthropomorphized computer helped the teacher discuss concerns specific to programs and their interaction with the programming environment. Presenting programming language as “computer speak” allowed the class to understand specific meanings associated with syntactic aspects of programs, such as { } and ;. Although trivial, these notional machines emerged as a response to student questions about the visible structural aspects of programs. This is salient, especially given that most students were programming in a text-based programming language for the first time.

Notional machines adopted within this unit not only differed based on the layer of abstraction they were communicating but also on the granularity of the concepts being simplified. Notional machines communicated the transformation of individual symbols, statements, blocks of statements, or the entire program during runtime into observed outcomes. Presenting only circuit-related dynamics without any connections to program execution was also noted, especially in the case of the first two curricular projects that involved only simple circuits. Examples of notional machines of symbolic meanings include describing “=” as a variable “getting a value,” which then leads to reading a variable definition statement like “int led = 10;” as “led getting a value of 10.” Further, notional machines such as “Microcontroller pins acting as programmable gates” allowed the class to reason light pattern blocks, or entire programs where the only function of the program was to make light patterns. Overall, notional machines in this unit were intended to help learners make meaning across the different parts of the program as relevant to the concepts explored during the curricular unit.

In addition to their themes and granularity, notional machines also differed in forms or representations. They took either the form of verbal explanations or roleplays that included students as participants. Explanations were narrations of the phenomenon under investigation, but in simplified forms. For example, emphasizing how a computer read a variable definition
statement ("int led = 10;") took the form of Ben repeating reading it in a way he expected his students to understand (e.g., "led is getting the value of 10").

Ben also employed analogies to phenomena outside the computing realm to communicate program behavior. Presenting an analogy between the runtime behavior of digitalRead() statements and people reading books was one such example. Performing short skits was another form where certain ideas were further highlighted through dramatization. For instance, Ben showed how variable names interact in a program through a skit. Roleplay was another form notional machines took during the unit. It involved students and the teacher taking particular roles and playing them out to understand program execution.

Overall, the analysis being one of the first attempts to study notional machines in practice, illuminated a three-tier framework to capture the different characteristics of these pedagogical tools (see Fig. 18). First, parallel to the themes discussed above, one of the key characteristics of notional machines is the level of abstraction of the concept—hardware, interaction between hardware and software, or software—that they are simplifying. Second, the specific mapping onto the program text can reveal the level of granularity of the concept. This is comparable to Schulte’s (2008) BLOCK model for student program comprehension. And, third, attention to the form of representation of the notional machine allows further examination of the nature of enactment of notional machines and the role of the teacher and the students.

**Figure 18: A representation of the three-tiered framework to capture notional machines in practice.**
Focused analysis of videos of four class periods further illuminated the multimodal nature of teaching with notional machines. The teacher was observed communicating ideas not only through spoken language but also via embodied gestures, use of on-screen representations, and voice modulations. For instance, Ben used his index finger to trace the direction of current flow within circuits, and spread out his arms to indicate the direction of signal when executing statements such as digitalWrite() that interact with the circuits. These enactments, when associated with programs, also involved specific on-screen representations and gestures. For instance, in episodes focused on the interaction between programs and circuits, Ben had the program on the shared screen, highlighting every line sequentially, while discussing the implication of its execution on the circuit. Furthermore, he stressed on key ideas by modulating his voice. He emphasized on parts of his communication either by slowing down or using a higher tone, and switched voices to indicate change of perspectives. This further adds nuances to our understanding of teaching with notional machines.

Learning with Notional Machines

The second group of questions were regarding student interactions and learning with notional machines. Particularly, around how students demonstrated conceptual agency while making sense of program dynamics during the unit. Conducting interaction analysis of eight class videos and analyzing student-generated artifacts through sociocultural theories of learning highlighted that learners were adopting and questioning notional machines, while making their conceptions visible and revising them. Vignettes from classroom lessons and their analysis highlighted the mediating role played by notional machines as students made meaning of program dynamics with e-textiles. Learners adopted prior notions discussed in the class to develop new notions that could explain newer programs, and revised notions as needed. They further adopted these notions to predict outcomes, and at times even questioned how certain notional machines could describe the system behavior. For instance, as shown in Activity 1 in Chapter 4, learners adopted “variables as remembering values” (anthropomorphized computer) notional machines previously
discussed while programming with digital inputs. At the same time, when learners noted inconsistencies between their previous understanding and the ongoing conversations, they asked clarifying questions and revised their notions. Roleplaying program execution further provided them ways of understanding the relationship between programs and circuit elements. In sum, although earlier accounts of the teacher's notional machines presented them as static, fixed computing models, student interactions with them during the unit highlight the different roles learners took in interacting with them in agentic ways and even causing the teacher to revise the notional machines.

Further, adoption of the sociocultural framework shed light on the situated nature of notional machines and their role in mediating and sustaining meaning-making within computing classrooms. Notional machines provided functional ways of understanding abstract and invisible program dynamics for the community of learners within the classroom. As emergent from analyzing classroom activities that were a month apart, learners adopted these notional machines as approximations to continuously expand their notions of program dynamics within the unit. Particular language and gestures had accrued special meanings among learners that constantly mediated their meaning-making of program dynamics. Rather than limiting student learning as mental models developed by individual students, sociocultural perspectives shed light on learners' meanings generation about program dynamics, their interactions, and agentic navigation of problem spaces within a computing classroom.

Students' Conceptual Shifts about Physical Computing Systems

The final question about capturing learning outcomes from the unit was: How did high school students’ conceptions of computing systems and programs shift after learning with notional machines? To answer this, ten students’ pre- and post-unit responses, which describe the structural and behavioral aspects of two computing systems each time (i.e., e-textile toys similar to their curricular projects, and everyday sensor-based computing devices like soap dispenser and smartwatch) were qualitatively analyzed. Students were presented with the functional
description of each of the artifacts and asked to comment on its structural makeup and underlying behavior that causes the outcomes they notice. First, this analysis revealed that students mostly had naive conceptions about the inner workings of computing systems, both real-world and project-like artifacts, before the unit; most students barely mentioned any computing, especially while discussing the real-world sensor-based automatic soap dispenser. Even when students guessed that computing or programs were automating or controlling the system, they barely drew out details, either in their structural or behavioral explanations. However, by the end of the unit, a significant number of students were able to discuss the structure and behavior of both these systems in more sophisticated ways, mentioning and articulating these aspects during runtime in greater detail. For instance, many more students identified different parts of the underlying circuits, separating sensors, actuators, microcontrollers, and the interconnections between them. They also articulated different programming constructs, specifying conditional control statements and potential logical expressions, and how that caused observed outcomes. All these findings, in addition to data that showed how students developed notions about program dynamics, highlight the potential of notional machines to not only help students understand similar computing systems but also systems that were not specifically discussed earlier in class, such as everyday sensor-based devices.

Developing conceptions of inner workings of devices, like sensor-based smart watches and soap dispensers, can potentially move students towards questioning and critiquing technologies around them and their designs. Notional machines, while deepening understanding of computing systems, can expand to include real-world devices and contexts spread across space and time. Learners can extend their understanding to discuss computing devices such as soap dispensers that discriminate against people with dark-colored skin, or smart watches that inaccurately work for dark-skinned and/or obese people. In today’s time and age when computing devices are inheriting, persisting, and amplifying societal inequities and injustices, it is important for computing classrooms to discuss the social issues and how they may interact with technical designs. Findings from the student interviews, although limited to a small group, shows the
potential to extend conversations within computing classrooms, particularly while discussing underlying program dynamics, to include their implications on marginalized individuals, communities, and societies.

Limitations

Despite promising findings, the study was limited in a few different ways. First, Ben, the teacher, had a longstanding relationship with the school, the ECS curriculum, and the e-textiles unit. He came with significant teaching experience and had been teaching high school computing for over 13 years in the same school. In addition, he was one of the first teachers to adopt the e-textiles curriculum and provide feedback to the curriculum designers about 5-6 years ago. Ever since, he has significantly contributed to the design of role playing lessons and revisions of the curriculum for online adoption during the pandemic. In sum, he holds a very agentic position when teaching this particular curricular unit. Ben’s extensive experience teaching a particular curricular unit, and even co-designing with research teams, is not representative of the broader trends with computing teaching, which has only been recently adopted within K-12 classrooms. While engaging Ben about notional machines may have been easier for similar reasons, future work with diverse teacher groups has a potential to address this limitation and shed light on teaching with notional machines.

A second limitation was the context of the school. Ben was situated in a charter high school that was particularly known for its strong STEM pipeline. For example, Ben himself had set up the Department of Computer Science within the school that provided an array of computing courses, starting from Exploring Computer Science to Advanced Placements Computer Science. This meant that although students who took ECS were mostly programming for the first time, they identified with computing in some ways and thought of the ECS course as either the first of the many computing courses they may take while in the school (most students in the class were freshman as an evidence) or had some specific interest in applications of CS (making games, making e-textile toys, etc). This was also visible in student interviews (the part not discussed
above). Teaching computing in a non-charter school, or a school without a pipeline of courses to offer, may significantly reshape the uptake of notional machines within classrooms.

Third, this study was conducted in an online setting due to the ongoing COVID-19 pandemic. The class met on Zoom, which offered very limited ways of capturing student presence. Only students who turned on their video, audio, or engaged in the chat were visible during the class. With the ongoing pandemic, the teacher and the school desisted from mandating any kind of participation (e.g., compulsory video on). This challenged data collection of students who did not or could not participate actively on Zoom. Unlike physical classrooms with videos that captured student presence anyway, data collection within online settings resulted in very few students leaving consistently rich data traces. This may have been particularly visible in the activity accounts in Chapter 4.

Fourth, students were physically distanced, taking classes from their individual homes during the pandemic that made collaborations and interactions among peers difficult. Although the teacher encouraged students to establish informal ways of staying in touch with each other by exchanging phone numbers and social media handles, student interactions through any of these were not visible to the teacher or me. In order to mitigate these to an extent, the teacher adopted Seesaw, an online platform for students to share projects in progress and engage with each other by leaving comments and likes. However, very few students utilized this option to engage with one another; rather, they treated the platform as a portal to “submit” their project progress like “class assignments.”

Fifth, the context of the pandemic and lack of in-person engagement with students also led to a much smaller subset of students who consistently participated in the interviews. Unlike prior years in Ben’s classes, close to one-third of the students and their parents did not consent to participate in the research study (10 out of 34 of them or their parents said no to research). Among the students who agreed to be part of the research, many of them barely turned on their videos or responded through chat, bringing down the students with pre- and post-unit matching
interviews to ten. Further, although the original study planned for a mid-unit interview, the online setting made it hard to conduct interviews within a comparable time frame.

Discussion

Overall, this thesis is one of the first studies examining key questions around teaching and learning with notional machines within an introductory high school computing classroom (Fincher et al., 2020). The in-depth video analysis not only illuminates rich accounts of teaching with notional machines, but also provides a framework for future efforts to capture notional machines in practice. It highlights the different levels of abstractions, granularities, and forms they take in practice.

Glass-boxing inner workings of computing systems, particularly program dynamics, through notional machines, had multifold opportunities to support students to: interact with program dynamics while reading, writing, and debugging programs in the immediate future within the curricular unit; and, extend their conceptions to similar devices in real-world contexts outside of the curriculum. Moreover, detailed analysis of learner interactions with notional machines highlighted their role as mediating tools and how they provide opportunities for learners to develop conceptual agency within computing classrooms. Given the increasing use of technology globally, there will be a possible surge of biases that will get embedded in the designs of everyday devices. Students within K-12 introductory classrooms not only need to be introduced to computing as tools to make personally meaningful artifacts that express their ideas but also to help them critically engage with devices around them and question how programs cause the outcomes they see. Findings emerging from the analyses presented in this thesis show promising roles of notional machines in glass-boxing computing within high school classrooms. The following section discusses these findings in relation to the existing literature on understanding teaching and learning with notional machines, and future research directions.
Glass-boxing Computing with Notional Machines

This thesis provided one of the first accounts from classrooms where a teacher taught an introductory computing high school class with notional machines (Fincher et al., 2020). This is in contrast to accounts of notional machines gathered in previous studies through instructor interviews after teaching, limiting such studies to instructor self-reports and/or their recollection of the most salient experiences with notional machines after teaching (e.g., Fincher et al., 2020). This study captured notional machines in practice by collecting and analyzing observational classroom video data that allowed for richer accounts of notional machines in practice and the different themes that instructors may adopt. These results call for more such studies to understand the role of notional machines within computing teaching.

Situated in a setting different from undergraduate and graduate classrooms where notional-machine-related studies have historically been conducted, the themes along which notional machines were taught indicated the key role played by contextual factors such as the programming environment. Although anthropomorphizing computers and microcontrollers connects to earlier examples from teaching programming with LOGO, notional machines such as Programs as Plays or Microcontroller pins as programmable gates were almost unheard of in prior accounts (e.g., see Fincher et al., 2020 for a catalog of examples). In addition, the different notional machine themes that the teacher adopted throughout the unit highlight how notional machines can be designed to sustain meaning-making over a unit’s duration instead of being “atomic” or “episodic” as discussed in other studies. Given that most computing courses and units are designed to build on concepts, from basic to advanced, this finding is not surprising. Since this was one of the first studies to explore notional machine usage over extended periods, such as a unit duration (Fincher et al., 2020; Guzdial et al., 2019), the thematic connections between notional machines as adopted by the teacher in this study calls for further studies to understand how notional machines are related across periods in other teaching-learning contexts.

Moreover, this was one of the first accounts of notional machines from a high school class teaching with physical computing (Krishnamurthi & Fisler, 2019). The context of e-textiles,
which lays out the microcontroller-based circuit, its components, and interconnections, led the teacher to adopt notional machines across different layers of the system and, at the same time, *glass-box* various program dynamics across different levels of granularity. Each of these levels correlated with Schulte’s (2008) BLOCK model to understand program reasoning. At the same time, they were connected to outcomes in the form of physical artifact behavior or processing during runtime. Particularly, the physical computing context presented a unique environment where programs could be discussed in terms of concrete materials, such as conductive thread and LEDs, and their interactions with program logic during execution. With physical computing on the rise within introductory K-12 settings (e.g., Kafai et al., 2019), future research with notional machines within these contexts should study the specific contributions of materiality and how teachers draw on them while engaging learners in program dynamics. Further, investigations can also look at the nature of notional machines within more modularized, tangible programming environments like robotics, which is another programming context popular for introducing K-12 learners to computing across informal and formal learning contexts (DesPortes & DiSalvo, 2017).

Analyzing notional machines in practice also highlighted the different forms, such as role plays, which literature barely has any accounts of but are hypothesized to exist in practice (Fincher et al., 2020; Sorva, 2013). At the same time, analogies and metaphors adopted within this classroom connect with other examples cataloged from undergraduate instructors teaching introductory classes (e.g., Fincher et al., 2020). More studies based on observational data collection and analysis could include diverse pedagogical approaches and notional machine forms that educators may adopt. Although notional machine studies in the past significantly focused on tool and interface designs, more recent interest and expansion of computing within K-12 classrooms call for future studies to look at teaching within these settings. Such studies can further illuminate aspects associated with notional machines in practice, such as multimodal teaching. Although not studied as notional machines, earlier analysis of computing teaching (e.g., Flood et al., 2018; Kwah, 2013; Solomon et al., 2020) had shed light on the crucial role played by teacher gestures and representations in communicating meanings about program execution. This
study not only confirmed the role played by gestures and representations, but also added another dimension of voice modulation. More studies in the future should explore the possibility of the teachers’ use of voice to draw students’ attention to specific programming aspects. Unlike earlier studies that had only considered the conceptions that teachers present, i.e., the computational model that they support learners with (Guzdial et al., 2019), closer look at the enactment accompanying notional machines reveal the multimodal nature of teaching.

Notional machines have been historically studied within undergraduate and graduate computing classrooms. But, this study highlights their affordance for computing teaching and learning within high school classrooms, even in introductory settings. This agrees with Fincher and colleagues’ (2020) hypothesis that notional machines are indeed ubiquitous within computing pedagogy and that it is just a matter of exploring them. Understanding the notions communicated within computing classrooms, especially introductory high school classrooms, becomes important since these notions significantly shape how concepts are made accessible to novice programmers and how they develop conceptions about underlying dynamics. Overall, findings from the analyses conducted in this thesis highlights the role of notional machines in glass-boxing computing across layers and granular levels and strengthens the call to study notional machines in practice within the introductory K-12 computing settings.

**Computing Learners as Meaning-making Agents**

Studying student interactions with notional machines highlighted the different roles students took while interacting with notional machines during the unit. Drawing from earlier studies, students were thought to just adopt instructors’ notional machines (Fincher et al., 2020). In part, guided by cognitive theories of learning, earlier studies were limited to collecting product data (i.e., student responses to surveys and think-aloud interviews) and analyzing them through similar frameworks to capture student learning with notional machines. However, this thesis also captured student interactions with notional machines in practice through sociocultural theories. It not only provided detailed accounts of student interactions but also over the course of the class unit. Video
recordings of class periods across a unit duration led to studying learner interactions from a historical and cultural perspective, treating the classroom as the community engaging in the practice of meaning-making of program dynamics within the ECS e-textiles unit. Unlike previous studies (e.g., Tunnel-Wilson et al., 2018), findings of this research showed that students took on agentic roles while working with notional machines (Fincher et al., 2020). Over the course of the unit, learners continuously adopted notions, questioned them, explained their thinking, revised them, and roleplaying program execution. Further, this analysis shed light on the situated nature of notional machines—socially constructed, historically accruing meaning, and produced within the context of the curricular unit and programming context. This finding strengthens Reisbeck’s comment about notional machines as being “locally useful” for learners to make sense of program dynamics within a context (Guzdial, 2017, p. 11).

This insight shifts our understanding of how students learn with notional machines. Understanding of student interactions with notional machines, especially through a sociocultural perspective, captures many of the concerns raised about studying student learning with notional machines, such as the role of programming languages, paradigms, and modes of engagement to name a few (e.g., Krishnamurthi & Fisher, 2019). Studying notional machines in practice and student interactions across different contexts will only further our understanding of their role in computing learning, and the influences of different contextual factors in shaping it. While not undertaken in this analysis, it may reveal how students draw from their social and cultural contexts while interacting with notional machines, especially when discussing analogies as another form of notional machines. These affordances of notional machines for student computing learning, although have been hinted at in prior studies (Guzdial, 2017), have not been explored enough due to a lack of studies examining notional machines in practice. This research, therefore, provides early insights into the ways in which notional machines can be studied to highlight student engagement with them within classrooms.
Expanding Computing Learning to Understand Everyday Devices

A part of the findings from analyzing student pre- and post-unit interviews with e-textile projects was not surprising, given that the teacher specifically focused on supporting students’ conceptions of inner dynamics of e-textiles. Among the ten students who consistently participated in the interviews, there was a considerable shift towards developing a more sophisticated understanding of the underlying structure and behavior of e-textile projects. Student pre-unit interview responses are in line with findings reported in prior studies that have highlighted the naive conceptions that students have about computing projects before any formal teaching (e.g., Cederqvist, 2019; Pancratz & Diethelm, 2020). However, with no study comparing student responses before and after formal teaching within classrooms, this study is one of the first accounts of how computing learning with notional machines can shift student conceptions.

One promising aspect of the findings from student interview analysis is how their conceptions shifted about real-world sensor-based technologies such as smart watches and soap dispensers, even when they were not discussed in the class. This confirms earlier hypotheses that learning computing with physical computing environments will allow students to make connections with similar real-world technologies (Przybylla & Romeike, 2014). Students, in this study, not only made connections but developed advanced conceptions about the invisible processes underlying these devices. Although earlier studies had noted student difficulties in uncovering the underlying behaviors of everyday devices (Pancratz & Diethelm, 2020), learning computing with notional machines within a physical computing context, particularly with e-textiles, seems to help learners mitigate these difficulties. For instance, the role of analogies drawn during the class to discuss program dynamics need to be studied to understand their connections to student conceptions of such devices.

While this analysis was limited to understanding students’ technical conceptions, particularly the structural and behavioral aspects of physical computing devices, future studies can look at how students bring in social and cultural aspects in relation to these technologies that
they can connect with (Kafai et al., 2019). For instance, what does it mean to develop conceptions about smart watches in relation to their differential outcomes for certain groups of people such as people with dark skin or with high body fat (e.g., Costanza-Chock, 2020). In such explorations, the definition of notional machines can be extended beyond technical program outcomes to include implications of differential outcomes to individuals and people. As learners engage with conditional statements and processing of sensor data when explaining smartwatch behavior and structure, computing classes can also support students to understand what those technical aspects mean to how devices interact with communities and societies. Such extension makes room for not only expansive learning (Greeno & Engerström, 2012) within computing classrooms, connecting computing across time, space, and people. It also opens room for critical engagement with computing and its role in the broader society, highlighting the invisible interaction between computing and societies (Kafai et al., 2019; Ko et al., 2021).

Conclusions

Overall, the analysis of teaching and learning with notional machines in practice has expanded accounts from K-12 classrooms, particularly teaching with physical computing construction kits. Further, video analysis has added nuances beyond drawings such as embodied gestures, voice modulations, and on-screen representations. Analysis of student learning with notional machines while interacting with notional machines highlighted learners’ agentic interactions with them while making meaning of program dynamics. Student responses articulating the inner workings of computing devices alluded to potential adoption of notional machines to not just deepen computing learning but expand them to contexts outside the walls of the computing classroom.
APPENDIX

A. Day-wise focus during the e-textiles unit

<table>
<thead>
<tr>
<th>Dates</th>
<th>210326 – 210328 (1-3)</th>
<th>210330 (9)</th>
<th>210332 (10)</th>
<th>210336 (11)</th>
<th>210338 (12)</th>
<th>210322 (13)</th>
<th>210324 (14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Circuit-based first two projects, Mood Lamp &amp; Wristband (NPD)</td>
<td>Introduction to Programming, interaction between circuits and programs</td>
<td>Making light patterns</td>
<td>Introduction to Debugging</td>
<td>Programming with a single digital switch input</td>
<td>Setting up student working groups (NPD)</td>
<td>Introduction to Mural project as a storytelling opportunity (NPD)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dates</th>
<th>210426 - 210428 (15-16)</th>
<th>210428 (17)</th>
<th>210430 (18)</th>
<th>210432 (19-20)</th>
<th>210422 (21)</th>
<th>210422 (22)</th>
<th>210426 - 210514 (23-30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Designing &amp; Crafting the Aesthetics of the Mural Project (NPD)</td>
<td>Storyboarding light patterns for Mural project (NPD)</td>
<td>Writing and understanding user-defined functions (NPD)</td>
<td>Programming with two digital button inputs</td>
<td>Wrapping up Mural Project (NPD)</td>
<td>Debugging pre-designed scenarios given in the curriculum</td>
<td>Designing buggy projects and debugging each other projects (NPD)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dates</th>
<th>210518 (31)</th>
<th>210520 (32)</th>
<th>210524 (33)</th>
<th>210526 - 210530 (34-36)</th>
<th>210502 (36)</th>
<th>210604 (37)</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Introduction, Designing, and Crafting Human Interaction Project (HIP, NPD)</td>
<td>Programming with an analog sensor</td>
<td>Programming and Testing sensors (NPD)</td>
<td>Working in groups on planning HIP program, testing sensors (NPD)</td>
<td>Programming HIP, including sensor value ranges (NPD)</td>
<td>Discussing and questioning sensor-based systems (NPD)</td>
<td>--</td>
</tr>
</tbody>
</table>

* NPD = No visible Program Dynamics interactions

B. Semi-Structured student pre-unit interview protocol

Preamble:
Thank you for participating in my study. I will be in your class, throughout this unit, to learn about how you learn!!
Today, I will be asking you a few questions. And, I will return back with similar questions during and after your unit to see how you think. It should not take more than 20 minutes each time.
Please know that whatever you share here will be only between us and will not affect your grades in this class either. These questions are not to judge you or your answers but just to see your thinking process. So, the more you tell me what and how you think, the better it is.

Here we go:
There are three parts to this interview:
Let’s start with your relationship with technologies and programming

1. Programming-related literacy perspectives:
   1. Do you like programming? Have you programmed before? In what form?
   2. Why did you choose this class?
   3. Why do you think someone should learn programming?
   4. What do you hope to get or learn out of this class? Anything specific about this upcoming unit?

Now, I am going to ask you about a few technologies that we use on a daily basis and then about some programs. Let’s start.
1. Have you ever used an automatic soap dispenser?
   a. What do you think they do?
   b. What are all the things you know about how they work?
   c. If you open apart this machine, what do you think you will see? What all may be there? [feel free to use the pencil tool to draw]
   d. What do you think each of these parts do?
   e. How do you think they work together?

2. Here is a plush toy that a student in another class made a few years back during this e-textiles unit [only the picture provided]. The student made this so that when someone presses both the wings lightly, the lights at the eyes blink and a song plays. But, someone presses both the wings harder, the lights at the eyes stay on and a different song plays.
   a. How do you think this toy works?
   b. If you open up this toy, what do you think you will see? What all may be there? [feel free to use the pencil tool to draw]
   c. What do you think each of these parts do?
   d. How do you think they work together?

[OPTIONAL—In case the student doesn’t sound familiar about either of those above artifacts]
Here’s one more technology that you may have used: automatic doors at super markets. Have you?
<automatic doors at super markets>
   a. What do you think they do?
   b. What are all the things you know about how they work?
   c. If you open apart this system, what do you think you will see? What all may be there?
   d. What do you think each of these parts do?
   e. How do you think they work together?

[OPTIONAL—In case the student doesn’t sound familiar about either of those above artifacts]
Here’s one more technology that you may have used: automatic lights. Have you?
<automatic doors at super market>

a. What do you think they do?
b. What are all the things you know about how they work?
c. If you open apart this system, what do you think you will see? What all may be there?
d. What do you think each of these parts do?
e. How do you think they work together?

[OPTIONAL—In case the student doesn’t sound familiar about either of those above artifacts]
You have seen stop lights at the junctions, right?
a. What do you think they do?
b. What are all the things you know about how they work?
c. If you open apart this system, what do you think you will see? What all may be there?
d. What do you think each of these parts do?
e. How do you think they work together?

That’s it. Any other thoughts you have?
Thank you so much!
I hope it was not too stressful!

C. Semi-Structured Student Post-unit Interview Protocol

Preamble:
Thank you for joining me today for one last time before we wrap up with this unit!

Just like last time, I will be asking you a few questions. It should not take more than 30 minutes.

Please know that whatever you share here will be only between us and will not affect your grades in this class either. These questions are not to judge you or your answers but just to see your thinking process. So, the more you tell me what and how you think, the better it is.

1. First of all,
   1. How did you feel about this class?
   2. What are 1-2 key things you have learned in this class that you didn’t know before?
   3. What are 1-2 things that you started thinking about differently because of this unit?
   4. What were 1-2 fun parts of this unit?
   5. What were 1-2 hard or challenging parts of the unit?

Thanks for sharing!

Now, I am going to ask you about technologies that we use on a daily basis and then about some programs. Let’s start.
2. Have you ever used heart rate monitors or pulse oximeters, or have you seen others use it?
   1. What do you think they do?
   2. What are all the things you know about how they work?
   3. If you open the device apart, what do you think you will see that is making it report the heart rate? [feel free to use the pencil tool to draw]
   4. What do you think each of these parts do?
   5. How do you think they work together?

3. Here is a toy similar to other e-textiles projects. It is made so that when someone presses both the hands lightly, the lights at the eyes stay on and it Foxy sings a piece of melody. But, someone presses both the hands harder, the lights at the eyes blink and a different piece of music is played.
   1. How do you think this toy works?
   2. If you open up this toy, what do you think you will see? What all may be there? [feel free to use the pencil tool to draw]
   3. What do you think each of these parts do?
   4. How do you think they work together?

[OPTIONAL—In case the student doesn’t sound familiar about either of those above artifacts] Here’s one more technology that you may have used: automatic doors at super markets. Have you?
   <automatic doors at super markets>
   1. What do you think they do?
   2. What are all the things you know about how they work?
   3. If you open apart this system, what do you think you will see? What all may be there?
   4. What do you think each of these parts do?
   5. How do you think they work together?
4. Before we close, I had a few questions about these questions I asked you during these interviews throughout this unit.

1. How did you like thinking about automatic soap dispensers, automatic faucets, and now, pulse oximeters during the unit?
2. Did you think about these while you were in the class?
3. When? And, in what form?
4. Thinking about what all we did during the class,
   a. Do you think anything we discussed or did particularly helped you learn about e-textiles better?
   b. Did anything that we did or discussed help you learn more about devices like soap dispensers or automatic faucets around you and how they work?

Any questions you have for me?

D. Codebook for notional machines analysis

<table>
<thead>
<tr>
<th>Notional Machine Themes</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthropomorphized Computer</td>
<td>Ben explains it as &quot;Because there were syntax errors, because things were not right, that things were not written the way that the language is expecting things to be written because the language is very, very stupid... we have very, very specific things that we have to write for the computer to understand. So, for example, PinMode is a very, very carefully crafted word is a function that the computer understands. OUTPUT is a specific word that the computer understands [highlights OUTPUT in the code]; HIGH and LOW [highlights them]. So we don't have on and off in computer speak. We have HIGH, LOW. digitalWrite is another. These are our keywords that, you know, they have very, very specific meaning to a computer. And if you spell it incorrectly or if you capitalize something that shouldn't be capitalized, the computer will be like, oh, wait, you're telling me something that I don't know about. I don't know what that is.&quot; (210310)</td>
</tr>
</tbody>
</table>

| | Ben draws upon an analogy to further explain the role of variables, this time taking a situation from daily life: "So, like, imagine what would happen if, like, I'm walking on the street, I met Mona, like, who are you? And she's like, I'm Mona. And I'm like, oh. And then I go back to my friends and I'm like, Hey, I met someone. And they're like, Who did you meet? And I'm like, I met Mona. And they're like, Oh, cool. Mona. Sweet. What if I'm walking down the street and I meet Mona. But I say, hey, what's your name? And she's like, you don't talk to me. And she walks away. Then I go back to my friends and I'm like, hey, I met someone, they'd be like, who'd you meet? I'm like, I don't know. I don't know whom I met..." Ben continues to expand on this idea of variables as naming/labeling values: "So we named things so that we can refer to them again later. [gesturing later as going back and pointing at something]" (210416) |
| Programs as Plays | Ben invoked “programs as plays”: “can someone want to tell me what character should I create? [cursor at the empty first section of the code] What components am I working with here? How do I tell the computer that the wristbands connected to pin 10?”; Harem typed in "led 10" and Dave "int led = 10"; Ben took the int part and typed it into the CB, while continuing to inquire about the variable name: "Do I have to call it led? Because what's currently being connected to pin 10?"; Mona, Karla, Harem typed in wristband, and Ben took that up in his code and completed that line of code "int wristband = 10;".... Ben got into the setup section [conveniently disregarding discussing the ; and even = in the first line but more iterating the "gets" semantic]: "The setup only happens once in the beginning of our code. Only happens once. And this is our characterization. [goes back to the metaphor of program as play and variables as characters in the play] And our characters can either be outputs or they can be inputs. (210312)

| Ben elaborates on the roles: "And so let me explain all the roles. So I'm going to be the processor. I'm going to read the code line by line out loud [gesturing the sequential execution by moving the hand in a sequence]. I'm going to tell all of you what to do. I'm going to make sure that each time that I run through this loop that my two buttons, button1 button2 have a different configuration than the last time I read. I want my actors to do different things each time. So as the processor, I'm also going to, like, make decisions just so that we have some diversity in terms of what you do. I don't want people clapping the whole time. That will be really, really boring" (210416)

| Anthropomorphized microcontroller | Ben tells that he's making the connection between how humans and microcontrollers process: "in order to determine whether the switch is on or off, we have to physically read it [gesturing directionality through hands]. We have to read the screen and interpret the screen, and a computer is no different. So here it is: digitalRead(switchy);. This is how the C.P.U reads switchy." (210318)

| Ben continues to expand on what's happening in the loop: "First, it reads the current, so it reads the sensor and then stores that value. In a variable called noise, remember that if I read something, I have to save it for later, so noise saving it for later." (210520)

| Microcontroller pins as programmable gates | Ben says: "Now we're getting there. Allowing electrons to flow through. Yes. And so if we allow electrons to flow through PIN 10, how are they getting to the LEDs? What's happening to the electrons? Where are they flowing?" He continues, after seeing students respond as alligator clips: "it would flow through the alligator clips, which now flows to your circuit on your wristband. Yes! Now. What do you think we're doing when we set pin 10 to low?" Mona suggested that less electrons pass through for which Ben asked if any pass through at all? Juan brought in a metaphor and said that "its closing a gate"; Ben appreciated and continued that "its just a switch except that it's a programmable switch.
PIN 10 is basically a gate. And we can open up the gate and allow all the electrons to flood through the gates, in which case the LED is turned on. Or we can close the gates. And now the electrons aren't flowing through and the LED is turned off." (210310)

Ben recaps how the switch states were understood and interpreted by the computer: "so with a switch, when the switch was high, it was connected, but when the switch was low, we disconnected it. [gesturing a open and close gate-like using his both hands in front of the computer] So there's this idea of on and off zero and one going on"; he tells "0 means cut off the circuit... and 1 means the circuit goes through"; he gestures open-close gate with hands in front of the computer to remind the class of open and closed circuit.. I'm really simplifying things right now, but at the core of the core of what's happening, it's a circuit" (210520)

Circuits as electron loops

Ben continues: "Exactly. Well, Mona says they're going to flow positive to negative [calling out the chat response]. So the idea here is if I energize pin 10, I can turn on the lazy river. And it starts flowing [hovering the cursor around the circuit drawing, drawing a loop around that line]. And if I turn off at 10. All the electrons are still there. But I turn off the lazy river. They're not moving anywhere. And the only way to turn on this LED is for them to pass through, to move through. So you've got all of these electrons in the Lazy River [doing the embodied gesture of lazing around in the tube] and they're all like inner tubes, drinking beer, like – hey Debra, like, you know, I don't know. I don't know what electrons do with our time. Who knows? Well, they're all just chilling out. Right. And then, like, we turn Pin 10 to HIGH." (210312)

Ben said “this is then connected to the processor on board and how it processes signals as a circuit loop [signaling the loop gesture using his index finger]” (210520)

E. Codebook for student conceptual understanding analysis

<table>
<thead>
<tr>
<th>Degrees of Sophistication in student explanations</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplistic</td>
<td>Structure: Input/output as the only components; no computation</td>
</tr>
</tbody>
</table>
"It, like, knows when something is under it. So, the sensor is like—okay there's something under there, let's let some soap out" (Karla, 210203)

"We can sew like lights and we have a metallic thread in that kind of thing. So, I'm thinking there's the buttons, there's two buttons in the wings, right...And they connect to the lights or they connect to a power source and then to the lights." (Raul, 210209)

**Behavior: Simple interactions between inputs and outputs**

"The motor spins around, probably spinning other components around that enables the lid to open." (Juan, 210201)

"I think kind of like our patch project on the circuit so that when you talk and stuff, so they could hear it. So, I guess the watch and the pulse oximeter, they can also feel your pulse too... I don't know how though" (Susan, 210525)

**Limited Structure: Acknowledgement of computational unit, without details and without recognizing interconnections**
<table>
<thead>
<tr>
<th>Behavior: Acknowledge behavior to be caused by some computation without specifics of nature or type</th>
</tr>
</thead>
<tbody>
<tr>
<td>“I think it's more of a motion sensor. So, it can detect when, like, when your hand is under it. Once it recognizes that something is under it, it disposes soap.” (Sasha, 210201)</td>
</tr>
<tr>
<td>“I learned this like outside of class, but you're able to tell your pulse if you put your fingers near your wrist or your neck and you're able to like a little you're able to feel the pulse. And yeah, I'm sure there's something in the in the watch and the pulse oximeter that's able to feel that and read it... this is entirely based off circuit playground” (Raul, 210527)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advanced Structure: Locate a computational unit; establishing interconnections between the inputs, outputs, and the computational unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>“there's a little box here and then wires that connect to the eyes, and then the box tells the eyes to light up when it's being pressed.” (Meena, 210526)</td>
</tr>
<tr>
<td>“And we know, it uses temperature and sound, sorry.. it uses light and uses sound to be able to do things. And so maybe we can connect to pressure or temperature or something like that.... like we learned in the circuit playground, like when it senses that it'll.. It's like relay to the screen or whatever, or the lights to display this, like if it was pulse was ninety-five, it'll tell like yeah, it's in ninety-five. So, display that.” (Raul, 210527)</td>
</tr>
<tr>
<td>Sasha, 210201</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>(Sasha, 210201)</td>
</tr>
<tr>
<td>(Sasha, 210201)</td>
</tr>
<tr>
<td>(Sasha, 210201)</td>
</tr>
</tbody>
</table>


155


Kafai, Y. B., Fields, D. A., Lui, D. A., Walker, J. T., Shaw, M. S., Jayathirtha, G., ... & Giang, M. T. (2019). Stitching the Loop with Electronic Textiles: Promoting Equity in High School Students' Competencies and...


