Teaching with Student Agency for Self-Efficacy when Learning PL Concepts across Experience Levels

A dissertation

submitted by

Matthew P. Ahrens,

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

August 2022

ADVISOR: Prof. Kathleen Fisher

Teaching with Student Agency for Self-Efficacy when Learning PL Concepts across Experience Levels

Matthew P. Ahrens
ADVISOR: Prof. Kathleen Fisher

Abstract In this thesis, we studied how Programming language (PL) concepts could extend beyond the classroom by improving self-efficacy with student-agency-focused teaching interventions in four case studies across three student experience levels. By letting first-time programmers choose songs they knew for programming musical performances in study one, Previously unengaged students collaboratively constructed knowledge with their peers regardless of programming experience level and every group successfully finished projects using the domain specific language (DSL) for event-driven, message passing programs that we made called BlockyTalky. However, instructor authority and desire for student elaboration provided unintended negative corrective feedback. By following a visual metaphor for dataflow programs in study two, first time programmers successfully designed wearable devices and implemented them using the dataflow DSL we made called Warble, as well as general-purpose, host language code. However, student satisfaction with simple designs reduced the computational concepts they explored. In study three, we had mid-level university students in an introductory PL course discuss begin recitations with explanatory feedback rather than corrective feedback. Perceived participation was not impacted, but this intervention warrants further study on early confidence in discussing algorithmic PL concepts rather than the new syntax and semantics we anticipated. In study four, a semester-long rubric enabled a student-regulated learning (SRL) assessment strategy for an upper-level programming language design seminar. Students achieved learning objectives through designing and implementing DSLs, often beyond instructor expectations, with negligible detriment to teaching effort. Students applied understanding in professional programming practice beyond the context of the seminar's learning environment.

Acknowledgments

This thesis work was made possible by support from family, the TuPL lab, faculty advisors for my academic and professional development, my committee, as well the participation and good nature of many student participants.
In no particular order, I would like to specifically thank JM, Karl, Lucia, Brian, Nate, Sam, Elise, Raul, Moses, Mike, Sara, Tomoki, Dylan, Janny, David, Ferd, Norman, Donna, Uma, Chadi, Torben, and Seikyung.

Matthew P. Ahrens

TUFTS UNIVERSITY
August 2022

Contents

Abstract iii

Acknowledgments iv

List of Figures viii

Chapter 1 Introduction 1

1.1 Context . . . 1
1.2 The goals of this dissertation . . . 2
1.3 First-time programmers . . . 2
1.3.1 BlockyTalky study . . . 3
1.3.2 Warble study . . . 4
1.4 Mid-level university students . . . 4
1.5 Upper-level university students . . . 5
1.6 Operationalizing student agency . . . 5
1.6.1 Theory . . . 6
1.6.2 Application to the studies . . . 7

Chapter 2 Background 12

2.1 Theories of Cognition . . . 12
2.1.1 Reviewing cognition . . . 12
2.1.2 Distributed Cognition . . . 13
2.2 Computer Science Education . . . 14
2.2.1 Study Focuses . . . 14
2.2.2 Epistemology for Science . . . 14
2.2.3 Epistemological Messaging . . . 15
2.2.4 Assessment . . . 16
2.2.5 Rubrics . . . 16
2.3 Programming Languages . . . 17
2.3.1 Computational Thinking . . . 17
2.3.2 Block-based Languages . . . 17

Chapter 3 Glossary 19

Chapter 4 The effect of first-time programmers' project design choices on PL concept learning and self-efficacy **24**
4.1 BlockyTalky . . . 24
4.1.1 BlockyTalky design . . . 24
4.1.2 BlockyTalky implementation . . . 25
4.1.3 PL concepts of BlockyTalky's computational model . . . 26
4.2 Case Study: Musical Performances Workshop . . . 27
4.2.1 Activity Context . . . 27
4.2.2 Data set . . . 28
4.2.3 Theory . . . 28
4.2.4 Small group work . . . 29
4.2.5 User story activity discussion . . . 47
4.2.6 Design Documents . . . 48
4.2.7 Design document summary . . . 53
4.2.8 Resulting code . . . 53
4.2.9 BlockyTalky discussion . . . 53
4.3 Warble . . . 55
4.3.1 Warble: a technical introduction . . . 57
4.3.2 Computational Model . . . 59
4.4 Case Study: Poster Design Activity . . . 59
4.4.1 Context . . . 59
4.4.2 Data set . . . 60
4.4.3 Poster artifact . . . 60
4.4.4 Resulting code . . . 62
4.4.5 Survey Data . . . 66
4.4.6 Warble Results . . . 67
4.5 Discussion . . . 68

Chapter 5 Self efficacy and participation in introductory PL recitations **70**
5.1 The course . . . 70
5.2 The issue . . . 70
5.3 Goals of the intervention . . . 71
5.4 Contributions . . . 71
5.5 Theory basis . . . 72
5.6 Methodology . . . 73
5.6.1 Participants and course context . . . 73
5.6.2 Handout and Experimental Design . . . 74
5.6.3 Survey Design . . . 76
5.7 Results . . . 78
5.7.1 Data Summary . . . 78
5.7.2 Choice of metrics for participation . . . 78

5.7.3 Choice of metrics for discussion preparedness . . . 79
5.7.4 Statistical analysis method choice . . . 79
5.7.5 Participation . . . 80
5.7.6 Task Preparedness Ranking . . . 81
5.8 Discussion . . . 84
5.8.1 Interpretation of the results . . . 84
5.8.2 Threats to validity . . . 87
5.8.3 Future work . . . 88
5.8.4 Conclusion . . . 89

Chapter 6 Student agency via self-regulated learning in a DSL design seminar 90
6.1 Our course . . . 90
6.2 The problem . . . 91
6.3 Our contributions . . . 91
6.4 Study-specific Background . . . 91
6.4.1 Student regulated learning . . . 91
6.4.2 Fostering participation . . . 92
6.4.3 Knowledge construction between students and instructors . . . 92
6.4.4 Where can we help TAs? . . . 93
6.5 Course Design . . . 93
6.5.1 Assessment and grading . . . 93
6.6 Results . . . 98
6.6.1 Qualitative: Student `META.md`s over time . . . 98
6.6.2 Quantitative: Statistics on class work . . . 98
6.6.3 Student Feedback . . . 100
6.7 Discussion . . . 101
6.7.1 Instructor experience . . . 101
6.7.2 Student Experience . . . 102
6.7.3 Changes to future iterations . . . 103

Chapter 7 Conclusion 104

Bibliography 108

List of Figures

4.1 A typical BlockyTalky program and its hardware and protocol configuration . . . 26

4.2 The finished user story poster for Bodhi, Carson, Delaney, and Avery with their four user stories and an illustration conveying the content of their performance. . 33

4.3 The poster containing the user stories and promotional flyer art of the group with youth leader Owen and youth participants Kelsey, Marley, and Lane 38

4.4 The end of activity poster with user stories from youth leader Jude and youth participants Harper, Ellison, and Finley. 46

4.5 The design document to outline the message passing computational model for the project of youth leader Bodhi with youth participants Carson, Delaney, and Avery. 49

4.6 The design document to outline the message passing computational model for the project of youth leader Owen with youth participants Marley, Kelsey, and Lane. . 51

4.7 The design document to outline the message passing computational model for the project of youth leader Jude with youth participants Harper, Finley, and Ellison. . 52

4.8 A sample of a synth program from group three where the motif extends for more than 60 more play note blocks. 54

4.9 An example, simplified Warble diagram showing dataflow between two hardware components parameterized by which hardware port they connect to. In text, the equivalent Warble DSL provides an implementation with added detail about the internal state of each component. 56

4.10 The design poster for a smart umbrella as a wearable program to be implemented with Warble. The participants made an illustration to help solidify the specifics of the project. They highlighted hardware components with a square box on top of the umbrella and a phone abstraction. A bubble and arrow diagram shows an intended universe for their program with the components labeled and connectors connected with types. 61

4.12 The finished code for the smart umbrella wearable project with a DSL quasiquoter containing the code corresponding to the participant's design diagram and the host language Haskell function `control` the students had to write to implement the behavior. 62

4.11 The design poster for a wearable that lights up when a button is pressed and displays the time. The participants made an illustration and attached the hardware of three components to it: a button sensor, an LED display actuator, and a crayon representing a laser pointer. The bubble and arrow diagram shows an intended universe for their program where two sensor and actuator systems are connected in parallel. 63
4.13 The finished code for the laser watch wearable project with DSL quasiquoters containing the code corresponding to the participant's design diagram and a custom component. 65

5.1 For just handout A (No intervention), the rank averages and percentiles for each task over each semester 82

6.1 An example student matrix of rubric items (learning objectives as columns, levels of understanding as rows) with cells representing the assignment on which the student demonstrated proficiency shown in the format $nn - m$, where nn is the class number and m is the assignment number. 93
6.2 An overview of a sample homework opening showing the rubric items suggested students earn on that assignment. 97
6.3 A comparison of how many students submitted project iterations and `META.md`s for each assignment (19 max). 99

1

Introduction

1.1 Context

This dissertation studies how students retain Programming Language (PL) concepts beyond their initial learning environment. We use the term far transfer to refer to this retention and generalize its definition as "human transfer of thinking and reasoning from one context to another" [BC02]. PL concepts include computational models that can be applied across many application domains as well as the decomposition of programming language features into syntax, semantics, and usage patterns. This dissertation uses the domain-specific language literature's definition of domain. Under this definition, application domain means an area of computing where programs share some context, such as assumptions, methodology, or technical dependencies [Hud96].

As an illustrative example, consider an introductory computer science course asking students to implement a recursive function for sorting a list of integers. The application domain of this function would be an educational context to illustrate a general purpose programming language feature; we often call programs in this context toy examples where there is no specific outer project or purpose for the function. Instructors might assume implementing the function correctly means the student can use recursion in other classes, in other programming languages, and with other constraints and cognitive loads (for example, on a whiteboard in a technical interview) [BLM+18].

While CS educators often use the correctness of code artifacts to measure how effectively students apply newly learned concepts, far transfer correlates with metrics like self-efficacy [HQ03]. Self-efficacy is a theory of one's belief about their own abilities among four categories. This dissertation focuses on students' belief in two of the categories: performance outcomes and vicarious experiences Performance outcomes relate to successful demonstration of accomplishment and self-belief in proficiency [BS81]. In CS, students are told they are proficient when they write code that meets expectations of correctness. Vicarious experiences relate to the "transmission of competencies and comparison with the attainments of others [...] and social influences that one posses certain capabilities" [Ban97]. In CS (and other STEM fields), we aim to foster positive vicarious experiences where learners collaboratively construct knowledge in a supportive learning environment., However, "both students and teachers in higher education are highly socialized

by the time they meet in the classroom, mostly in traditional learning environments characterized by communication behaviors that lead to defensiveness rather than supportiveness" (for example, unfounded assumptions and competitiveness between learners) [GB04].

To observe self-efficacy, this dissertation analyzes non-code artifacts like design documents, surveys, and reflections as well as how students discuss concepts in small groups. Using non-code artifacts as a communication medium, students express intermediate states not captured by the finished work [BSD14]. The understanding students have when communicating contains information about their self-belief of their proficiency. In service to vicarious learning, non-code artifacts often contain explanations and questions that construct a shared understanding [BF06].

In our studies, we chose which non-code artifacts to collect based on how disruptive they would be to the learning environment. For example, in a highly customizable learning environment with no grading expectations, student group discussions can be fully captured on video as a non-code artifact without negative impact to the student's well-being. In university classroom environments with grades and assessments, we indirectly abstract information from typical coursework to reduce observer effects and survey fatigue [PWW04].

To encourage student self-efficacy in PL learning contexts, this dissertation focuses on teaching interventions that support the ability of students to make choices about how they learn. Designing their own projects or authoring their own assessment and acceptance criteria are examples of these choices. While overloaded in the education literature, this use of student agency focuses on a definition with respect to student motivation: "persisting and acting on choices to reshape the environment" [Vau21]. Generally, any choices students make to relate their learning to a context not provided by an instructor exhibits the student's agency.

1.2 The goals of this dissertation

We hypothesize that there should be evidence of beneficial far transfer outcomes at all experience levels when teaching interventions encourage student agency. Our analysis illustrates effects on students' self-efficacy from clinical uses of student-agency-focused teaching interventions across student experience levels. We expect improvements to student self-efficacy, specifically belief about proficiency and vicarious experiences amongst peers, from increased student agency in small group learning environments. We believe PL concepts, specifically, benefit from improved far learning outcomes as far transfer of PL abstractions and decomposition improves computational thinking in other application domains [Rin08]. To investigate how student-agency-focused teaching interventions improve far transfer of PL concepts through self-efficacy outcomes, this dissertation presents four case studies across three student experience levels: first-time programmers, mid-level university students, and upper-level university students.

1.3 First-time programmers

Chapter 4 presents two studies for teaching first-time programmers with domain specific languages (DSLs) and student-agency-focused teaching interventions. Risks to student agency in-

clude communication breakdowns amongst group members or with the instructor as well as time and scope mismanagement [KLT22]. These risks restrict student agency when one student does a disproportionate amount of work due to their technical skill, when design decisions overfit to instructor feedback, and when decisions are made based on what is easy due to criteria external to the project.

By having first-time programmers design programs in a DSL's restricted application domain, we anticipated lower barriers for peer communication, program progress without excessive instructor guidance, and quick iteration through limited PL expressiveness. Since many complex programs could be made in the application domains, we anticipated positive perception and outcomes of using the DSLs by first-time programmers despite evidence of negative perception of the professionalism of block-based syntax amongst high-school learners and uncertain lifetimes of DSLs [WW15] [DCBB15].

1.3.1 BlockyTalky study

In a summer camp for high-school aged youth, students created musical performance pieces using a block-based domain specific language (DSL) and programming environment we implemented called BlockyTalky. During a teaching intervention, students authored single-sentence user stories to structure their projects in the computational model provided by BlockyTalky. We observed students as they discussed answers with partners of various authority: peers, youth leaders, undergraduate and graduate instructors, and the entire group. We analyzed these discussions through the lens of a distributed cognition approach to qualify student self-efficacy as group knowledge with respect to the restricted application domain, the computational model, and the artifacts that represent both. We discuss this analysis in context with their final designs and implementations.

Students successfully made projects of their own design with BlockyTalky. The use of musical performances as an application domain enabled students agency to choose project content which encouraged students who had previously not participated during purely technical activities. By approaching misunderstandings of the computational model from concrete musical performance concepts, students were able to make progress and collaborate on knowledge without getting stuck on correctness at the design level.

However, students' non-interactive interpretation of the musical performance application domain restricted their exploration of message passing, concurrency, and distributed design. The design activity instructions introduced students as agents in the application domain as performers with the intention of considering ownership of sensors at runtime. For example, *who* would press the button that plays the melody. Students often made themselves the primary agents in discussions about the computational model and explanations abstracted away the sensor hardware and events at a high level when expanding the computational role of ownership using the word *control*. This ownership became overloaded as it referred to all of: performing a musical role at runtime, being responsible for writing the code for a particular musical role, and as a placeholder for the implementation that computes values. For example, *I will control the volume*.

1.3.2 Warble study

In a second summer camp for seven high-school aged youth, students designed and implemented wearable devices with a textual DSL we created called Warble. Students placed sensor and actuator hardware on a poster, drew connections between the hardware with function-labeled arrows, and drew the context the device would be used in. The goal of this activity was for students to design with full agency and not consider implementation difficulty while also restricting the activity with the dataflow computational model Warble provided. If successful, students could implement their designs by simply translating the diagrams to DSL code that syntactically matched those bubbles and arrows, annotating their programs with types, and then implementing small, host language functions to implement the transformation logic of arrows.

Students successfully made projects of their own design with Warble. As the primary benefit, students showed computational thinking through designing their projects in terms of types and data transformation using the visual metaphor with a clear idea of the project scope. As the primary detriment of the intervention, students avoided complex implementations if they could not readily come up with arrow labels or type annotations.

1.4 Mid-level university students

Chapter 5 presents a study with mid-level university students in an introductory programming languages course. We opened each of the weekly small group labs, called recitations, with critique activities on several existing, imperfect answers. In the existing structure, TAs often found that soliciting participation initially was difficult and that some students had low confidence in their ability to get started. We believe this contributed to TA frustration when referencing the pre-recorded lecture materials students were supposed to have watched and other ill effects during subsequent office hours. To more easily start discussions, we modeled the activities after how successful TAs said they responded to minimal initial participation. When receiving no volunteers for participation, TAs would present how they, themselves, model the problem, e.g. visually on shared media like a whiteboard. Then, they would ask students to comment on differences between the model and the student's own understanding.

Our intervention A/B tested answering the first recitation question for corrective feedback or critiquing existing answers for explanatory feedback. We administered weekly surveys that measured perceived participation and self-preparedness to quantify changes in self-efficacy without burdening students with additional responsibilities.

The introductory activity being changed to critique for explanatory feedback did not lead to noticeable differences in participation. The intervention did show students have higher self-belief in preparedness to discuss theory heavy topics rather than ones that introduce new syntax, especially in an in-person learning environment rather than an online one.

These results do not give us a definitive answer to how to help TAs more effectively start discussions, but they help us prioritize what topics from the introductory PL class deserve more attention for student agency, namely advanced computational models using existing syntax and semantics like continuation passing style using higher order functions. In other CS domains, like

algorithms, decoupling the concrete application from the abstract theory helped educators focus efforts on improving self-efficacy and soliciting student questions [EK17]. We would expect similar benefits from applying further interventions to these more difficult, conceptual topics in introductory PL.

1.5 Upper-level university students

Chapter 6 presents a study where we taught an upper-level programming language seminar in which students learned how to read and discuss research papers on Domain-Specific Languages (DSL) and designed, implemented, and presented prototype DSLs in domains of their choice as semester-long projects. We designed a two-dimensional, semester-long assessment rubric listing the content areas in which students needed to show proficiency and the levels of understanding at which they needed to show each. As a student-agency-focused grading strategy, we credited proficiency in a given rubric cell whenever a student demonstrated it. To support this assessment model, we structured a blended-learning environment; students completed reading, video lecture, and implementation as online assignments. Small group discussions and labs allowed students to practice their learning in class. At the end of class, students produced documentation in the form of wiki summary entries summarizing that class session's discussion. Students referenced this documentation in their reflection entries, and we instructors used the documentation to develop homework, such as what topics needed more examples and what the DSL design evaluation criteria should be. Along with homework assignments, students produced reflections which they used as the assessment instrument and we used to determine the needs of future class activities. This feedback loop formed our integration between the online and in-class components.

Students successfully implemented DSLs of their own design in application domains of their own choosing. Students achieved far transfer oriented learning objectives at increasing levels of understanding often beyond instructor expectations with little impact on grading or teaching effort. Students made connections to the application domain as it exists in the wild and to general programming practices which supports our goals that the learning will be re-applied outside the context of the class' learning environment. From these results, future research can improve the learning environments for earlier experience levels by scaffolding the most successful student agency activities like reaching out to code library authors, connecting the work to other CS courses, and presenting details of special topics in addition to the PL concept accomplishment we instructors find valuable.

1.6 Operationalizing student agency

In the four case studies this dissertation presents, student agency is an independent variable instructors control through the design and facilitation of teaching interventions. This subsection describes a framework for how student agency was controlled and how it manifests in the teaching interventions for the purposes of understanding the studies and generalizing the approach.

As an assumed context, the default teaching intervention design is not student-agency-

focused, which we describe using the characteristics of *instructionalist* teaching interventions and *well-defined problems*. To describe the student-agency-focused teaching interventions we designed for each study, we construct *ill-defined problems* by negating these characteristics and packaging them in design and critique activities. Lastly, we describe for each study where we *encouraged* students to take control of their learning using a construct called *framing agency*, and how we planned to observe how students *actually exercise* that control using a construct called *epistemic agency*.

1.6.1 Theory

Instructionalist teaching interventions and well-defined problems We assume a default context of *instructionalist* teaching interventions using purely well-defined problems "which consist of a well-defined initial state (what is known), a known goal state (nature of the solution), and a constrained set of logical operators (known procedure for solving)" [Jon00]. Instructors know the answers to well-formed problems making the instructionalist approach both easier and more predictable to facilitate. From prior experience, students may believe the problems given must be well-defined depending on the conventions of their discipline [TYC12] [SMC+09].

Ill-defined problems The four case studies presented in this dissertation employed student-agency-focused teaching interventions. More specifically, the teaching interventions provided opportunities for students to make choices about one or more unknown aspects of solving a problem. When constructing these teaching interventions, we replaced a known aspect of the solution to a well-defined problem with an unknown aspect to form a *ill-defined problem* [Eas69]. An aspect of problem solving is unknown if it includes any: *elements* that are not known, e.g. by the instructor [Woo83]; *multiple solutions* or no solutions [Kit83]; *multiple valid criteria* for evaluating solutions [Kit83]; or *uniquely human activities*, i.e. expressing judgments or beliefs [ME89].

Framing agency In learner-friendly language, instructors often present ill-defined problems as design or critique problems [RO+90]. However, facilitation and assessment difficulty, as well as student inertia from expecting well-define problems, leads instructors to provide superficial choices instead, i.e. student choices that let the problem still be well-defined with each aspect known. Our four studies present teaching interventions where learners made consequential (non-superficial) choices over their learning which is known as *framing agency* [SP20].

Epistemic agency Learners then must act on these opportunities. Learners acting to build knowledge can include, but is not limited to, motivations for learning, for modeling practice, for intrinsic value or interest, and for real-world application. The construct of these motivations is called *epistemic agency* [MMR+18]. After observing the data and results, each chapter discusses the epistemic agency that students exhibited to qualify how the intended framing agency manifested in student behavior.

1.6.2 Application to the studies

The studies in this dissertation were performed with a variety of learning environments and experience levels as opposed to highly-controlled laboratory experiments with less variation. Due to this variety, the experiments could not contain the same unknown aspects to control. To help compare the approaches, this section presents an overview of what framing agency was intended for each study. To summarize, table 1.1 shows framing agency as three qualitative variables describing when a student could make a choice about problem framing: initial state, logical operators, and goal state. We describe framing agency using these three variables from the structure defined for well-formed problems. The initial state provides the assumed knowledge the student and the instructor has, the logical operators provides the set of steps the student must take to solve the problem, and the goal state provides the end result the student would recognize as a solution. To show exactly how we add agency to this framing, i.e. well-formed problem structure, we choose whether the initial state, logical operators, or goal state will be well-formed or not. Specifically, each variable can have either the value *known* or can have one or more values we derived from the unknowns that make up ill-formed problems: *elements*, *solutions*, *criteria*, and *judgment*. We define these values as:

- *Known* means an instructionalist approach was taken for that part of the problem framing.
- *Elements* means that the instructors did not have knowledge of parts of the problem framing, i.e. students were expected to become experts to solve some aspect of the problem.
- *Solutions* means multiple valid solutions were believed to exist (these studies did not offer opportunities with no solution), i.e. students could provide answers other than ones the instructor expected as satisfying solutions.
- *Criteria* means there were multiple ways to define the acceptance or satisfying criteria, i.e. the student would tailor the aspect of the problem to what they believed was correct or useful.
- *Judgment* means there was a uniquely human aspect to solving the problem, i.e. the experience and beliefs of the student determined what a satisfying solution would be.

For example, if a teaching intervention allowed for judgment in the logical operators, the specific steps the student takes to find the solution depends on their experience or beliefs such as choosing whether to use the methodology modeled by the instructor or one from a previous class to solve a problem.

Study	Initial State	Logical Operators	Goal State
BlockyTalky	Known	Solutions, Criteria	Solutions, Criteria
Warble	Known	Solutions	Solutions, Criteria
Introduction to PL Handout A	Known	Known	Known
Introduction to PL Handout B	Known	Known	Criteria, Judgment
PL Design	Elements	Solutions, Criteria, Judgment	Elements, Solutions, Criteria

Table 1.1: The operationalization of the qualitative variable *framing agency* and its setting per study.

1.6.2.1 BlockyTalky agency

In the BlockyTalky study, the teaching interventions intended framing agency where the goal state and the logical operators contained unknowns. The workshop defined the goal state as a musical performance, but provided no technical detail as to what a *good* performance would look like or must contain. Students were encouraged to choose what music and actions their performance would consist of. To differentiate these student choices from "menu options" [SP20], where superficial choices would not impact learning, the students made choices for the end goal state to determine the abstractions for their project, the amount of complexity or detail, as well as what a satisfactory musical performance must contain. Specifically, even after choosing a song, many concrete project implementations may satisfy "perform song X with BlockyTalky", but students got to decide what *satisfy* meant to their groups.

The logical operators expected to use the BlockyTalky DSL and toolkit in the activities during each workshop section, but there were no explicit set of instructions for how to build the project nor prepare the musical performance. Students could choose, reject, or modify any aspect of the activities when applying the learning to their performances. As a tinkering approach [VB14], students were asked to deconstruct existing examples for reusable pieces (i.e. blocks or pre-combined collections of blocks) and decide whether they wanted to use those pieces in their projects wholesale, modify them (e.g. change parameters), or request new pieces from instructors using the lack of appropriate existing pieces as evidence of their need. For example, this approach resulted in new block abstractions being designed and implemented by instructors during the camp due to student request.

The initial state was assumed as known since the workshop began with activities ending in deterministic outcomes for all students. The collection of all example projects approximates a start state, and instructors assumed no prior programming knowledge from any of the participants. From these assumptions, all students were put through the same instructionalist learning approach to learn the language features of BlockyTalky until the student-agency-focused teaching interventions were used for musical performance project design and implementation. However, in the application domain of musical performances, some students did have more prior knowledge than others which affected their choices of non-code logical operators like with what resources they could use to transcribe the music into BlockyTalky code, e.g. sheet music, piano tablature. Since the teaching interventions did not explicitly facilitate this choice as a starting point and instructors had expert musical knowledge to provide to students, the initial state for all students was

considered to be known.

1.6.2.2 Warble agency

In the Warble study, the teaching interventions intended framing agency where the goal state contained unknowns and the logical operators had a known acceptance criteria, but unknown content parameters like size and complexity. The workshop defined the goal state as a wearable project the students would present at the end of the camp. Like the BlockyTalky study, the satisfying criteria that defined a good, finished, or fully-featured project was left to the students' creation. Finished projects, like a mood ring that used a heart rate sensor to change the color of an RGB LED, provided an example of what the instructors thought a finished, wearable project could look like. The goal of providing this example was to show the hardware the students would be using, to model the diagram students would use as a visual metaphor to design their projects, and to preemptively prevent projects with infeasible constraints, i.e. the example exposed the hardware and was presented as a functional *prototype* rather than a finished, commercial-grade product.

The logical operators were fixed in their form; students had to encode their ideas as bubble-and-arrow diagrams modeling dataflow computation, and then students translated those diagrams into code using a formulaic process. These instructions limited students design ideas to only those that could be expressed in that form. However, there were no restrictions or instructions on how many components (bubbles) or computations (arrows) the students had to use, nor on how complex (connectedness) the diagrams had to be. Students were asked to determine these quantitative features of the diagram based on what they thought would get them to their desired goal state of their projects. In exchange, projects designed encoded as these diagrams were assumed to be easily translatable into DSL code.

Similar to the BlockyTalky study, the initial state was known as the workshop began with activities ending in deterministic outcomes for all students. Students were taught the basics of programming using Warble as well as algebraic data types, primitive values, and functions in the general-purpose host language Haskell. Students were assumed to have no prior programming knowledge nor any knowledge on the implementation of wearable devices.

1.6.2.3 Introduction to PL agency

In the Introduction to PL study, the teaching interventions intended framing agency where the goal state contained unknowns. The Introduction to PL course provides instructionalist assignments asking students to follow the same steps towards a deterministic goal. In our study, the first question of the recitation activity had the modest intention of not having an intended correct goal state. That is, while the question presented in the playscripts had a correct answer, there was no intended correct discussion the students were expected to have. Students were asked to discuss what they liked, did not like, agreed with, and disagreed with. From these instructions, there are multiple satisfying and even contradictory satisfying answers such as one section discussing part of the playscript positively while another may discuss the same part as negatively depending on the students' personal judgment. Both of these approaches would be correct from the instructors'

point-of-view since the goal was for students to assert a position, recall the corresponding lecture material as a shared start state, and assume a critic's role of authority over the material.

The logical operators were fixed as the length and mode of the discussion were provided by the instructors. Students were not able to guide or design aspects of the discussion itself. I authored the playscripts, chose what material they would cover, and TAs facilitated the discussions in a standardized way. Alternative approaches that could provide framing agency at this step would be to have students choose which sub-topics from the lecture materials and to what degree they would discuss different playscripts based on their own needs, but as the remainder of the activity had specific content goals that depended on knowledge from the first question, letting students express this agency was not feasible.

The starting state was the same for all students. Specifically, the lecture materials and book were the resources students were instructed to use in preparation for the recitation activity.

1.6.2.4 PL design agency

In the PL Design study, the teaching interventions intended framing agency where the initial state, goal state, and logical operators all contained substantial unknowns.

Like the BlockyTalky and Warble case studies, the goal state was a DSL of each student's own design that satisfied criteria the students collectively authored. The teaching interventions provided scaffolding for students to abstract the satisfying criteria from common features of published academic DSL papers. As the aim of this scaffolding, the satisfying criteria would be concrete and available to the instructor for reference in feedback, but the students would have decided what makes a good DSL and what features were necessary and not necessary for their learning in the course, e.g. what features of DSLs exist in the academic papers, but would not be necessary for them to exhibit in their projects.

The challenges of this approach were that such satisfying criteria had to work for each student's project, which resulted in abstract, parameterized, or predicate guidelines in student authored class notes rather than as instructionalist assessment steps or grading with point values. From this abstractness, students needed to interpret how the satisfying criteria applied to their project, causing variation in effort depending on the difficulty of their chosen domains or the desirability of certain language features. The assessment rubric rewarded learning objectives exhibited rather than satisfying criteria met in part to avoid unfairness from these challenges.

The logical operators that students should take were provided in homework activities. However, the only guarantee the operators provided was that doing the suggested activities would exhibit the learning objectives. Similar to the example projects from the BlockyTalky and Warble studies, the goals of these homework assignments were to practice and understand examples that employed the suggested techniques. How the practice applied to each student's particular DSL was left to the student to discover; students could skip, add, or modify the homework assignment to fit their assessment or project needs. As an illustrative example, students were asked to write five programs for their domain of choice in Haskell. The instructionalistic goals of this activity included the amount of programs (five), the scope (a reasonably small program in the domain), and the method (using the Haskell host language). The agency-focused goals of this activity included

choosing which particular programs would provide insights into: the difficulty of the domain, the computational model the DSL would provide for the domain, what language features the DSL could provide, and what features of the host language the DSL would need to compile to. Towards these agency goals, a few students changed the instructions (choosing less than or more than 5). In making their own agency-focused goals, two students produced programs in multiple domains to help them choose which domain they wanted to proceed with. But most students followed the instructions as written, expended similar effort, and made progress in terms of the overarching assessment.

The initial state of the teaching interventions required students to choose a domain. As opposed to the "menu options" design choices where each choice is equivalent, but stylistically different, the choice of domain affects what particular DSL features, host language features, and independent knowledge students would need to learn about in order to meet the shared satisfying criteria of the end goal. Most critically, the instructors would not know elements of the domain or how they would manifest in the students DSLs.

Students chose their domains for a variety of reasons and often in opposite directions. For example, many students chose their domain because of prior knowledge and comfort, but some of these students chose domains they had low prior knowledge in to use the course as an opportunity to explore while others chose domains they had high prior knowledge in to focus on the DSL creation or the application outside of the course. Both of these choices are equally valid towards the learning objectives and could provide opportunities for students to exercise epistemic agency, but facilitating both requires the instructor to ensure fairness, which we achieved through our assessment structure, goal setting, and reflection.

2

Background

In this chapter, we frame our approach through the lenses of cognitive science, computer science education, and programming languages. Cognitive science foundations provide a vocabulary and methodology for us to observe the details of student interactions in small groups. We use the social cognitive theory of self-efficacy to analyze the observations which informs our understanding of the learner's behavior, environment, and self-belief. With these high level learning goals, we use computer science education research to guide our learning environment and teaching intervention construction. Lastly, we tailor this guidance to the specific challenges of teaching Programming languages concepts.

2.1 Theories of Cognition

Non-code artifacts provide a broad array of detail because they are often unstructured tools used by the learner to communicate understanding in intermediate steps or to summarize thoughts on an activity. To help analyze the non-code artifacts we collect, a theory of cognition helps the selection of interesting features as well as methods for how to explain the analysis of the excerpts. The primary theory of cognition chosen for this dissertation facilitates finding self-efficacy exhibited in the non-code artifacts produced in small-group contexts.

I adapted this description reviewing cognition and describing distributed cognition from previously published joint work that describes projects related to the BlockyTalky study presented in this dissertation. In this previously published work [DSA+15], I analyzed a portion of the video as well as authored diagrams illustrating the computational model corresponding to student and TA explanations.

2.1.1 Reviewing cognition

Most research on cognition in computing ultimately focuses on how *individuals* develop computational ideas. We often make assessments of students understanding from the end product, the program, they produce. Social [Vyg80], situated [ARS96; CBN89; Lav88], and cultural [Col98; Rog03] understandings of cognition and learning challenge this tradition. Research on learning shows extreme difficulty for learners to transfer knowledge from one context to another [Pac01;

BBC99]. From this evidence, our desire for far transfer encounters challenges from generic difficulty in addition to those specifically from learning PL concepts.

Different conditions (settings, problems, partnerships) activate different knowledge in learners' minds; learners frequently develop and demonstrate different, even contradictory, ideas about the world, and which ideas are operational at any moment is highly context dependent [SDR94]. In considering this model of how students learn, we focus on small-group activities to solicit, capture, and study their construction of the knowledge and intermediate states. In service to far transfer of programming languages, we often see this difficulty manifest in a preoccupation with syntax. As a PL educator, it is difficult since semantics and computational models are often transferable. But, this belief erroneously leads us to think that "all introductory courses teach the same things" (graciously interpreted as: fundamental computational thinking is universal) [LSA+18] and thus *syntax shouldn't matter*. But the impact of syntax is yet another example of how cognition is context dependent among other factors specific to PL [SS13].

The term *cognition* traditionally refers to a model of the brain's mechanisms for information processing, error correction, memory, perception, and communication. However, research on situated, embodied, and social cognition challenges this model by highlighting ways in which real-world cognition cannot be easily located solely within the skull. As an example, students who struggle with arithmetic in class can successfully make purchasing decisions using prices at the grocery store or in other *real-world* contexts like selling school candy [Sax88].

2.1.2 Distributed Cognition

Distributed Cognition [Hut95] (DCOG) theory generalizes this perspective on real-world cognition, arguing that all of the cognitive functions (e.g., memory) that have historically been analytically located in the head can also be seen in the emergent properties of interactions between people and tools in culturally- and historically-produced settings. In proposing a framework for studying "Cognition in the Wild," Distributed Cognition theorists highlight several key aspects of cognitive systems that researchers should attend to [HHK00]:

- The distribution of cognitive processes across social groups, including ongoing redistribution of activity to balance cognitive load.
- That culture intimately shapes cognition by offering tools, settings, and social norms through which to work or through which work can break down.
- Effortful coordination between internal (to the head) and external (tools and environment) structure, including through the use of the body of the problem-solver.
- The event-driven path dependence of cognition, where "the products of earlier events can transform the nature of later events."

DCOG has many strengths over traditional approaches to understanding cognition. Perhaps most powerful is its ability to account for how a diversity of tools and representations are key elements in socio-technical systems of problem-solving; specifically, that external tools and representations are not just objects used by people, but can do cognitive or social work, such as transforming one kind of information into another or prompting a group to talk about their knowledge in particular ways.

DCOG challenges researchers to take careful account of how individuals interact with one another, tools, and setting, and to observe the ways in which those interactions produce the cognitive functions of information processing, memory, sensing, error correction, etc.

In this view, *learning* is no longer just change in individuals' conceptual models or behavior, but also includes changes in the relationships between individuals and in their individual and joint relations to tools and settings, which can also be modified over time. For example, introducing a new tool (say, a DSL with a specialized syntax, computational model, and simulator runtime system) to a physics team that previously had been required to model free-body diagrams by hand is a form of learning: the team becomes able to do existing work faster and possibly also becomes able to take on newer, harder problems (through iteration and feedback from the tool). The group has learned, but that learning cannot be fully located in any of the individuals alone, nor in the individuals without their tools (who possibly might only be able demonstrate knowledge through the use of the tools). Rather, the learning consists in how their emergent system of interaction reconfigures itself to solve problems in new ways, such as by re-distributing cognitive load onto tools. Using DCOG in the analysis of learning can offer insights into how activity (including learning) happens that are not possible within the traditional cognition-is-inside-the-head paradigm.

In this dissertation, we consider the shared knowledge built by students along with how that knowledge manifests in the tools, the non-code artifacts, and the reflections they author to complete work that gets embodied as programs.

2.2 Computer Science Education

2.2.1 Study Focuses

A literature review of introductory programming research found that "the largest number of [introductory programming] papers reviewed focused on students, followed by teaching, then curriculum, and finally assessment" [LSA+18]. From these papers, the lack of context about the teaching environment most severely inhibited generalization of the results of these studies. As general advice, "Systematic studies are needed to further investigate whether the [teaching] tools and approaches actually result in improved outcomes for students, rather than simply providing additional information for teachers." Towards this effort, this dissertation presents a vertical view of student agency in activities for learning Programming Language concepts. The studies in this dissertation emphasize the activity content, student interactions during the teaching of the activities, as well as assessment and feedback.

2.2.2 Epistemology for Science

CS projects often benefit from iteration because it enables incorporating feedback and extensive testing. Students can learn useful skills by modeling the processes industry professionals use for these tasks [Bra03]. Blindly emulating these processes in our teaching interventions, however, can require students to forgo major learning opportunities in favor of wrestling with technical details

that do not significantly advance their learning. In a similar critique of teaching science in general, Russ describes shifting to an *Epistemology for Science*: specifically, that "justifications for valuing particular ways of thinking or doing in science learning would have to be based on whether and how such epistemologies are productive—both for learners and from the perspective of learners" [Rus14]. For first-time programmers, we often choose tools and curriculum content from this perspective: a priority of general-purpose language learning over computational thinking, using tools with a high learning curve, toy problems tangential to the core learning objectives, overwhelming IDEs with professional features, etc. In advanced courses, this problem manifests when research papers or materials that we expect students to learn from solve much larger problems than student projects would normally encounter, and student compare their own progress to these projects.

In designing curriculum for far transfer and self-efficacy, we must consider if our content and methodology includes technical baggage and bloat or provides usefulness beyond the classroom as a learning objective. For example, when considering block vs text for first-time programmers, a general-purpose language like C or Java satisfies professional practices, but includes sharp edges that may hinder introductory computational thinking. We can abstract away many of those sharp edges with block languages, but then suffer if applicability to real-word programming is a learning objective. This dissertation explores several teaching interventions and tools without believing one approach is superior, but we root these choices in the specific learning objectives for the learning environment of the study, which may differ depending on external constraints (grading expectations, institutional objectives, course dependencies, etc.)

2.2.3 Epistemological Messaging

An important subset of the social context for all the studies is the interaction between students and instructors. Instructors try to understand what aspect of an activity do students have questions about at any moment. In work on epistemological messaging [Rus18], Russ defines four categories of attention to student thinking:

1. Attention to the content of the idea
2. Attention to the form of the idea
3. Attention to the production of the idea
4. Attention to the relationships within and across ideas

Often in our assumptions, students should be ready to apply the topic presented in lectures or examples so that instructors can ask students to produce code or other artifacts demonstrating understanding of the topic.

For first-time programmers, instructors help students build shared vocabulary to use some computational model which often results in spending time on the *content of the idea*. For intermediate programmers, TAs assume students know the content and, instead, want to direct their attention to student thinking related to the *production of the ideas*. For expert students, instructors assume student independence and direct most of their attention *within and across ideas* where

students are asked to become experts and the instructor takes on the role of facilitating shared knowledge construction amongst the group.

2.2.4 Assessment

2.2.4.1 Bloom's Taxonomy

Many CS instructors have used Bloom's Taxonomy to design assessments for their course's learning goals. A commonly-used revised version of the taxonomy describes classifying educational objectives in two dimensions: The act students are being asked to do, i.e. *Remember, Understand, Apply, Analyze, Evaluate, Create*, and what thinking is happening in the student's mind, i.e. *Factual, Conceptual, Procedural, and Meta-cognitive* [Mar06].

However, researchers and educators have found difficulty in implementing the original and revised taxonomies: "They [both] fail in providing non-ambiguous definitions of cognitive processes and their categories. Moreover, their nomenclature is not always properly understood. Finally, no diagnosis function is available to practitioners, which should have been developed by the authors of the taxonomy or by experts of any discipline"[Vel21].

Due to these difficulties, educators often create assessments bottom-up from each individual assignment's acceptance criteria rather than top-down using a taxonomy. In the studies in this dissertation, we see how in-the-moment assessment of first-time programmers follows that acceptance criteria, and instructors make conclusions about student proficiency. However, discussion and TA observations offers perspective in terms of what the student is being asked to do and how they are thinking which leads us to different conclusions. We see how Bloom's taxonomy would also become complicated by the social aspects of learning in small-groups. With advanced programmers, we use a taxonomy of our own design to mitigate these difficulties, to communicate learning objectives to students, and to have students communicate their reflections back to us instructors. From this work, we see that viewing student learning through a shared taxonomy can be helpful, but the specific learning environment, experience level, and application domain complicates broadly applying a one-size-fits-all taxonomy.

2.2.5 Rubrics

Education research uses the term *rubric* as a place holder for a systematic assessment structure [WFC99]. However, "research that reports rigorous evaluations of rubrics might not state if the rubric is secret or shared; replacing feedback or complementing it; or how the rubric was explained to its users" [Daw17]. In formally communicating expectations to students, the studies in this dissertation explore how curriculum choices for student agency lead to the absence of formal rubrics in the first-time programmers level and the design of a accomplishment-based rubric at the upper level.

2.3 Programming Languages

2.3.1 Computational Thinking

Computational Thinking (CT) encompasses the educational goals of computer science that lives in the head of the learner separate from the tools and programs they create. Instructors are developing classes to capture these benefits and provide them to non-programmers with the expectation that they do not become programmers afterwards.

For example, in a CT course with big data, "What we offer is that the data and questions to be answered are decided by the student and are not predetermined assignments. Though we are using big data our goal is to teach computational thinking using big data as opposed to the goal of "data science" courses where big data is itself the object of study" [KBC15].

Goals in classes like these are echoed in reflections on how the study of programming languages affects computational and general thinking. "Programming languages has produced a larger, more precise, and more sophisticated range of abstractions than any other field. It is therefore the ideal context in which to build an understanding of abstractions. This understanding is a core prerequisite for a successful career in computer science. It can also be one of the keys to understanding unstated assumptions in other fields, which in turn opens the door to new ways of conceptualizing and eliminating limitations in these fields" [Rin08]. This assertion, specifically for the PL concept of abstractions, highlights the same benefits of computational thinking which is to apply a computational problem solving method to broader problems.

2.3.2 Block-based Languages

I adapted this description of block-based programming languages for first-time programmers from previously published joint work that describes projects related to the BlockyTalky study presented in this dissertation. In the previously published paper [SA16], I contributed the section on block-to-text interoperability and correspondence which describes my work designing and implementing the textual DSL language front-end for BlockyTalky.

Millions of students learning to program today begin with blocks-based systems. Blocks are visual representations of program code that add shape, color, and drag-and-drop editing to concrete syntax. Appearing in structured editors as early as 1992 [Min92], blocks offer a number of affordances: prevention of syntax errors, type hinting, and API discovery (block editors typically provide a palette of available objects and methods). Novices, like first-time programmers, benefit from these affordances to overcome some common challenges of learning to program, and there is modest evidence that blocks can improve student learning compared to text [Lew10] [Wei15].

Several research projects provide access to computing in various application domains with blocks-like PL interfaces. The Snap! Project investigates how blocks can scale up from simple to complex programs, such as those involving complex data structures, user-created abstractions, object-oriented programming, interaction with the Web, or even high-performance computing techniques.

Many programmers, including novices, are resistant to graphical interfaces that enhance the experience of programming: one study of entry-level high school programming students found

that while the students overwhelmingly perceived blocks-based programming as easier, they also saw it as less powerful, slower, and inauthentic[WW15]. African-American students' preferences for blocks or text vary sharply depending upon their long-term career goals, with those who identify programming as their career goal preferring text over blocks [DiS14]. Deciding on how to address when and whether we value these preferences provides a concrete instance of preparing teaching interventions in either the epistemology of science vs for science preferences.

At least for now, it is impossible to participate in existing socio-technical ecosystems of general-purpose programming without becoming fluent in reading and editing text, including text written by others. In an educational setting, new tools have been made to bridge gaps here like in assessment and feedback for The Snap! Project. [MCL+21]. Regardless of whether blocks become the next general-purpose standard, learners in the near future will need to transition from blocks toward text.

3

Glossary

The following is a non-comprehensive alphabetical list of terms we use in this dissertation, and their definitions as we use them. These terms may have broader definitions; however, we provide the particular definitions we believe are most useful to reading this dissertation.

Glossary

Abstraction In programming languages, an idea or computational concept providing a name, usage, and external functionality without consideration of implementation or internal behavior in a particular computing system.

Acceptance Criteria The assertions a learner uses to evaluate the representation of their learning. Learner's acceptance criteria include but are not limited to the correctness (sufficiently right or wrong), the completeness (sufficiently done or not done), and the specificity (sufficiently detailed or vague) of their representations.

Additive Grading A grading methodology where points or merits accumulate from positive demonstrations of understanding found in student work.

Application Programming Interface (API) An collection of abstractions useful for writing particular programs. APIs differ from DSLs by not providing their own computational model, but instead fitting the model provided by the host language.

Application Domain A categorization of computation by grouping programs that share aspects or purpose. Informally, the programs from an area of study or work with shared assumptions or context particular to that area.

Blended Learning Environment A classroom structure containing an online component to facilitate learning content discovery, an in-class component to model practice and social learning, and an explicit, purposeful integration between the two components.

Compiler A piece of software that translates programs from one programming language's syntax to another. For example, translating programs to a form that can run on a particular computing system with the original semantics preserved.

Computational Model A particular perspective to reason about and construct computation typically supported by syntactic and semantic choices of a programming language. Informally, a collection of programming language features intended to be used together to make styles of programs easier to write, read, and maintain.

Computational Thinking (CT) An approach of restructuring problems in terms of computing abstractions assumed to be solvable given representations that implement those abstractions.

CS Unplugged A teaching intervention design where students practice computational models and thinking without feedback from a computer system.

Distributed Cognition A theory of cognition where learning and knowledge account for the context and environment of the learner in addition to the internal thought processes that the learner experiences.

Domain-Specific Language A programming language providing a particular syntax, semantics, and computational model to better fit programs for a particular application domain. Informally, a programming language incorporating a domain expert's knowledge, assumptions, conventions, etc. into the design of the language.

Epistemological Messaging A high-level theory of pedagogical practice to direct instructors' focus to a student's particular stage of thinking about an idea. Specifically, considering if a student is thinking about the content of, form of, production of, or relationships between ideas.

Epistemology of Science A high-level theory of curriculum design that derives teaching interventions from real-world practices or professional behavior.

Epistemology for Science A high-level theory of curriculum design that prioritizes the efficacy of a teaching intervention more than the choices practitioners who use the concepts would make .

Grammar The lexicographic rules that define permitted syntax in a particular programming language.

Integrated Development Environment (IDE) A piece of software that provides a user interface for convenient usage of a compiler, runtime system, or interpreter for a particular programming language.

Interpreter A piece of software that executes programs written in a programming language directly, often as a read-eval-print-loop (REPL) style interface. Informally, software that iteratively compiles and runs programs in a particular programming language.

Just-in-Time-Teaching An iterative curriculum development methodology where feedback and observation of current student behavior influences subsequent content and teaching intervention design.

Knowledge (or Understanding) Construction An iterative process where a learner or learners present their understanding at intermediate states for feedback and reshaping. Informally, how learners test their knowledge to determine if they have successfully learned a concept.

Library A collection of representations, typically implemented code, intended to be directly reused in particular kinds of programs. For example, libraries are one method to implement and distribute APIs.

Mastery-Based Grading A grading methodology that recognizes achievement in overarching student behaviors rather than correctness of particular tasks on particular assignments.

Metaprogramming A language feature where the programming language exposes its compilation to the programmer. DSLs are often implemented as compile-time computation in host languages that provide metaprogramming.

Performance Outcomes One of four sources of self-efficacy that describes how proficient students believe they are at doing a particular task based on accomplishment.

Programming Language Feature An abstraction of a particular utility that a programming language provides. A subset of the language's syntax and semantics. An affordance of the programming language.

Programming Language(s) An area of study in computer science about the syntax, semantics, and usage of computer programs. A particular syntax, semantics, and usage for describing computation often including computational tools to compile and run those programs.

Representation In programming languages, an instance of a computational concept as an implementation in a particular computing system.

Runtime System (RTS) A piece of software specific to a programming language to execute the behavior of programs written in that language.

Self-Regulated Learning A pedagogical practice of making the student an active agent over their learning by facilitating and supporting goal setting and reflection phases in addition to production of work on concepts.

Self-efficacy A metric from social cognitive theory of a learner's belief about themselves derived from performance (accomplishment) outcomes, vicarious experiences, verbal persuasion, and physiological feedback.

Semantics The behavioral representation of computer programs in a particular programming language. What a program will do when run.

Subtractive Grading A grading methodology where a perfect score loses points or merits when negative demonstrations of understanding (such as mistakes) are found in student work.

Syntax The literal, lexicographic text representation of a computer program in a particular programming language.

Vicarious Experience One of four sources of self-efficacy that describes how students believe they understand and perform relative to the understanding and performance of others.

Visual Metaphor Ad hoc, tangible representations of computational thinking. Informally, diagram and playacting conventions that represent parts of a computational model to allow students to collaboratively construct and evaluate understanding while avoiding implementation details and compilation errors specific to code representations.

4

The effect of first-time programmers' project design choices on PL concept learning and self-efficacy

4.1 BlockyTalky

Aimed as a toolkit for youth to create computer music systems for collaborative music performance, the block-based domain-specific language (DSL) and programming environment we call BlockyTalky enables users to construct a variety of physical, sensor-rich interfaces. It supports the combination of different student-made technologies in distributed systems (e.g., sensors communicating wirelessly with software sequencers and synthesizers).

This introduction of BlockyTalky is based on collaborative, joint work that has been published in two papers [SKAF16] [SKA+17]. These papers contain descriptions of the BlockyTalky toolkit and an accompanying user study, but not the analysis presented in this chapter. As contributions to these works, I designed and implemented a textual DSL for BlockyTalky with compilation, its one-to-one correspondence *from* blocks, as well as compilation *to* a collection of targets including a message-passing based runtime system of my design, a wrapper for the Lego hardware library using a foreign function interface, and a DSL implemented in the Ruby programming language for musical applications called SonicPi. My contribution to these papers was to describe the summer camp as a contrasting experiment. The summer camp and its study are described in this chapter in more detail.

4.1.1 BlockyTalky design

Systems that integrate real-time sensing, programming logic, sound-making, and networked communication are quite common in computer music, yet existing programming tools designed for youth do not target this type of design. For example, Scratch is a popular software system for kids to use to create games, stories, and animations [RMM+09]. Scratch enables users to write programs by dragging pieces of code onto sprites in order to define their behaviors. Scratch allows users to play sounds and even to build modest physical interfaces for triggering sound (via

MaKey MaKey). However, the sound programming features are more suited to adding sound to games than they are to composing and performing music. EarSketch is another educational programming environment, and one that combines music with CS education [FMM+14]. It enables users to programmatically sequence and manipulate synthesized sounds and samples. However, it is not designed for collaborative real-time performances or for using physical inputs to shape sound synthesis.

BlockyTalky replicates certain successful aspects of tools like Scratch and EarSketch; for instance, their browser-based interfaces drastically reduce the complication of system use in schools. However, design and engineering patterns that are common in the design of new digital musical instruments fit BlockyTalky's computational model and language features, such as those featured at the International Conference on New Interfaces for Musical Expression (NIME) or the Guthman Musical Instrument Design Competition.. Such instruments can often be understood as assemblages of the following types of components: sound synthesis methods that offer real-time control over their parameters, sensing hardware that obtains real-time information about performers' physical actions, software that controls the parameters of sound synthesis in response to sensor data and/or algorithmic processes, and networking components that support distributing sensing and sound-making (and possibly also communication and synchronization between multiple performers). As engineers, we created a DSL and toolkit that enabled young learners and first-time programmers (aged 10 and over) to create these types of components and connect them to make new systems for live music performance. Furthermore, as educators, we wanted to empower users of this toolkit to learn about relevant computational concepts. For instance, BlockyTalky requires students to design their own high-level networking protocols—messages passed between different devices for synchronization and control. Students often must discover and adapt to constraints such as latency in sensing and concurrency. This is essential for students to learn as they design instruments that allow them to improvise during performances using physical inputs or modification of code. Furthermore, building instruments for live performance can lead students to discover how different configurations of sensors support different types of physical interaction.

4.1.2 BlockyTalky implementation

BlockyTalky is open source and runs on low-cost single-board computers like the Raspberry Pi. Typically, small expansion hardware attach to these boards to provide support for modular sensors and actuators, including child-friendly LEGO Mindstorms. Each BlockyTalky device runs a web server that provides users a complete user interface for configuring and programming its hardware. Drop-down menus enable users to declare what kinds of sensors are connected to each input port, and the system provides real-time sensor readings to help users plan, monitor, and troubleshoot their designs. A variety of programming blocks enable users to define musical motifs (including synthesized notes and samples), to send messages between devices, and to create event-handlers for inputs received from physical sensors or for messages received over the network. The programming model assumes that users will typically compose and enact performances using several physical devices at once, with some devices handling sound synthesis and others handling physical inputs from users; user-created asynchronous messaging protocols coordinate activity across

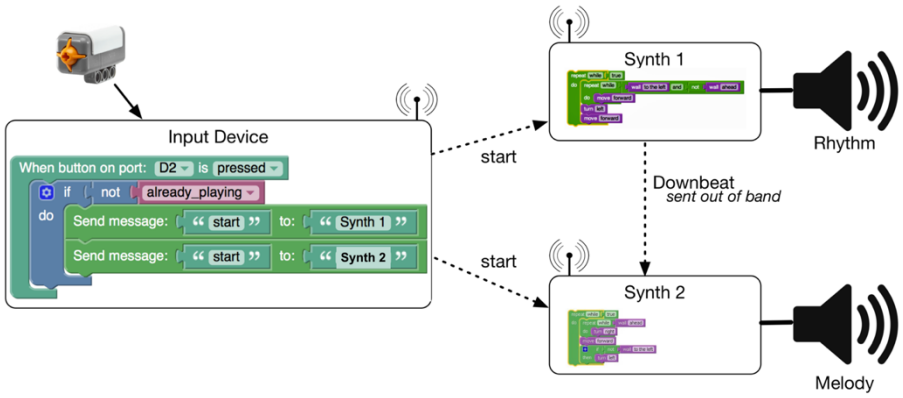


Figure 4.1: A typical BlockyTalky program and its hardware and protocol configuration

these devices. Figure 4.1 shows a typical configuration along with code for that configuration.

Users can configure BlockyTalky synthesizers to synchronize their clocks to one another by choosing one synthesizer to serve as a reference clock, and programming the other synthesizers to subscribe to that clock (using a *Sync to* block). Then they can use *Wait for* blocks to specify the timing of note synthesis or sample playback. For instance, multiple synthesizers can synchronize their actions by waiting for the same event, such as the next downbeat.

4.1.3 PL concepts of BlockyTalky's computational model

BlockyTalky's computational model contains two primary language features: event-driven program entry and message passing. Event-driven program entry provides multiple entry points to a program, as opposed to a typical beginner's program containing a single entry point like a main function. For example, a BlockyTalky program can contain an entry point for when the program starts running, for when a sensor value is over a certain threshold, and for when the program receives a particular message. Message passing in this computational model consists of one running BlockyTalky program being able to send messages to other running BlockyTalky programs. The intended use of message passing is for computing units, the BlockyTalky Units (BTU) to trigger events and start computation in each other using string identifiers (human-friendly names and messages) on a local network.

These BTUs are complete computing units as they run a web IDE for students to write, run, and debug programs in, and they use the BlockyTalky compiler and runtime system to run computation locally which minimizes dependence on external computers. Laptops can access the IDE by local address for convenience, but the devices themselves can connect to monitors, keyboards, and mouses to be programmed on directly.

As a form of abstraction, a BTU system is made up of mutiple BTUs running programs that send messages to each other by name and intended purpose. For example, a BTU with a button sensor could send a message to a BTU running a musical synthesizer so sound would actuate when the button is pressed.

As the intended benefit of this computational model, designing and implementing BTU systems lowers the barrier for multi-modal computing and protocol design. Multi-modal com-

puting better resembles the computational thinking that students typically see when using smart phone and web applications. To make these programs, the programmer must handle input from multiple sources and unify multiple APIs to design and implement a system's overall behavior. BTUs as individual, named, physical computing devices tangibly reinforces the abstraction provided by the computational model similar to how phones, laptops, and servers are components of a complex distributed system that logically equivalent programming activities do not capture. For example, multiple processes running on the same machine.

4.2 Case Study: Musical Performances Workshop

We ran a youth summer camp workshop for pre-university first-time programmers at an urban-area neighborhood center. Nine youth and three youth leaders participated to learn computation through programming musical performances. Three undergraduate, one graduate, and one professor instructor participated by designing activities and facilitating teaching interventions for creating musical performances with BlockyTalky. The workshop spanned eleven 90-minute sessions. Over the course of the workshop, participants worked in small groups of three youth and one youth leader to design, implement, and perform a musical performance of their choosing using BlockyTalky. Each group was given two BlockyTalky units: a sensor and actuator hardware unit and a synthesizer unit. They could ask for additional units if their projects demonstrated a need.

By the eighth day of the summer camp, instructors believed participants had a grasp on how the language features of BlockyTalk worked, but did not believe the groups had clear plans for implementing their particular projects, yet. Instructors led the groups through an unplugged design activity with the intent of moving towards concrete implementations.

4.2.1 Activity Context

We present this particular activity because students worked with their peers, the youth leaders, and the instructors to construct a shared understanding in terms of the application domain and the computational model. To accomplish this activity, each youth participant was given five cards. The cards were labeled U (user), V (verb), O (object), A (attribute), and S (sensor) where the students wrote their name on the U card and the other cards were populated with terms related to BlockyTalky and the application domain of musical performances. As an example of what to do with the cards, we used the cards U (Ellison), V (controls), O (volume), A (chorus), S (touch button) to produce a sentence we termed as a user story "Ellison controls the volume of the chorus with the touch button sensor". The intent was for participants to have a concrete sentence that they authored to help determine "what" to program to get started on their projects.

If participants did not like the story they had made for whatever reason, then they could trade with each other or with instructors who had a reserve of cards until they were satisfied. There were no intended correct answers for this activity and no acceptance criteria was given other than if participants "liked" their results. Once they liked their user story sentences, they were asked to write them on a large sheet of project paper.

Participants were given the additional instructions that, after they finished their user stories, they should diagram what their performance might look like (with stick figures, for example) as if it were an advertisement poster for their performance. The goal of including this detail was to encourage them to put the BlockyTalky hardware into their musical performance designs to promote thinking about implementation choices. The pedagogical worry was that elaborate project designs would encourage putting off implementation details and become too difficult to finish by the end of the camp.

When all the groups had finished, participants visited other groups' posters to offer feedback in the form of comments on sticky notes attached to the poster. By making the feedback a physical entity rather than verbal, we hoped it would be easier for the groups to offer, receive, and persist the feedback into the next session.

4.2.2 Data set

The data set for this particular design activity includes audio and video data from four cameras to observe each group, undergraduate instructor notes where each instructor primarily observed one group, and the finished poster non-code artifact. To put these results into context, this data set also includes a design document the groups authored during the ninth session to scope their projects in terms of BlockyTalky code and an example of the resulting code artifacts participants produced by the end of the camp.

From the full data set, we selected excerpts containing instances where participants constructed shared understandings and made design decisions by exercising their agency. By observing how the participants mention the computational model, we want to know how their word choice manifests with respect to different roles of listeners (peers, youth leaders, and instructors). Names have been psuedoanonymized to protect the identity of the individuals, but are used consistently between excerpts and analysis.

4.2.3 Theory

We conduct a Distributed Cognition (DCOG) analysis focused on knowledge construction related to the application domain of musical performances with the computational model of distributed, event-driver computation. In this study, the familiarity and rapport between participants and the role of the participants form a culture as a body of knowledge. For example, a youth participant demonstrates different cognitive behavior and expresses different understanding with an instructor participant than with their youth participant peers for the same computational thinking idea. Their understanding is not more correct in one interaction than another but, instead, together form a more complete picture of what the student thinks, how, and why. While the existence of a different manifestations of understanding between peer levels may seem obvious, highlighting interactions between different roles informs how BlockyTalky's design and our teaching intervention choices change exercised student agency and impact self-efficacy.

In previous work with BlockyTalky, analyzing non-code artifacts using DCOG provided perspective into how computing knowledge manifests with tools like the BlockyTalky hardware

and musical instruments like a guitar that would be unavailable purely in the mind of the learner [DSA+15]. The analysis in this study applies a similar approach focused on how participants express positive or negative self-efficacy with respect to the application domain of musical performances, explanatory and corrective instructor feedback, as well as PL concepts. The PL concepts for this study include the computational model, the application domain of the educational DSL, and connecting ideas between the two.

4.2.4 Small group work

This section on the small group activity presents transcribed excerpts from the video data with lines of dialog numbered, labeled with the speaker, and annotated with metadata from the video in square brackets. Below each excerpt is a constructed DCOG-aware, qualitative elaboration from the view of an instructor with expert knowledge about the application domain, computational model, and learning environment. From this expert knowledge and non-verbal information observed in the videos, we derived motivation for the dialog of each student.

We label each speaker with their peer level: *yp* for youth participant, *yl* for youth leader, *ui* for undergraduate instructor, *gi* for graduate instructor, and *up* for university professor.

4.2.4.1 Group one: Youth leader Bodhi and Youth participants Carson, Delaney, and Avery

```
1 Avery (yp): I want to control the.. [trails off] adjust from the card. so changes?
2 Parker (ui): say "Avery uses the ultra sonic to adjust the pitch and volume."
3 Avery (yp): [confirming] So, I'm controlling the volume.
```

Undergraduate instructor Parker assists youth participant Avery in completing the card activity. Parker knows both the correct form of responses to the unplugged activity and the intended goal of the activity: to scope the participants' projects into a collection of small, implementable user stories that fit with the computational model. Avery, like the other youth participants, knows only the instructions of the activity and the activity's relation to project planning: to make a sentence from the cards to help scope the performance. Youth participants were not explicitly given goals related to the computational model. There has been a mistake in the cards given to Avery as Avery has two attribute cards (pitch and volume), but this mistake does not impact this excerpt's analysis of how she constructs knowledge with Parker. Avery relies on the cards to decipher the activity. She interacts with Parker for explanatory feedback by asking questions related to her current understanding expecting answers in the form of examples or definitions rather than correct or incorrect. Avery trails off to inspect her verb card which is labeled *adjusts* (line 1) and introduces two verbs *control* and *change*. From an domain expert's perspective, the word control was introduced by the instructors during the activity instructions to mean a user performing a physical action with a sensor during the runtime of the performance. Change relates to the application domain, describes project specific computation, and is synonymous with adjusts without loss of meaning. Parker models the form of the expected response (line 2) using the language of the cards and not the synonym suggested by Avery where the verb *uses* is equivalent

to control. Avery confirms her understanding of Parker's help by summarizing (line 3). However, in doing so, she applies the verb control directly to the term volume instead of the sensor as modeled. Her intended meaning is ambiguous between the musical idea of volume in the runtime of the performance and the computational idea of the value of the volume state in the program. Both participants leave satisfied with the exchange of information. If Avery's question was about the production of the story then Parker's answer modeling the content, form, and production all at once provided an example of correctness in all of those cases. However, Avery and Parker may have an epistemological messaging mismatch with respect to what understanding was addressed from the interaction. Avery's summary reasonably describes the end-to-end purpose of the program Avery would make, but it abstracts away the role of the sensor and the application domain verbs *change* and *adjusts* from her original question.

```
1  Carson (yp): [Referring to poster with card stories] that makes sense, right?
2  Group: [no protest, little acknowledgement,
3          attention shifts back to laptops with piano transcriptions of songs on them]
4  Parker (ui): Now we are going to think about the final project flyer
5  Bodhi (yl): This is like the real thing now.
6  Group [attention directs back to listening to picked song for transcription]
7  Parker (ui): You could be controlling sensors as you sing.
8  Bodhi (yl): Or we could play the melody with the...
9               [gestures at a BlockyTalky Unit] ..the synth.
10 Carson (yp): Now that I'm writing it down, I'm scared
```

At this point in the activity, the participants have begun interacting with the cards and poster from the teaching intervention after both seeing the youth leaders and instructors model the process as well as asking initial questions. Youth Carson has added her user story to the poster and asks for feedback (line 1). The group offers little to no feedback as they interpret Carson's request as asking if her response contains anything that would make it incorrect by some unspoken criteria (lines 2-3). In this case, no feedback implies her answer is correct by the unspoken consensus of the group. Undergraduate instructor Parker shifts focus into realizing the project concretely on the poster (line 4). Youth leader Bodhi remarks a shift in attitude towards the project at this stage. From an instructor's perspective, this shift implies that there was a non-real or practice phase before this point in project planning (line 5). Instructors view the group's shifting of their attention to transcribing their chosen song as necessary, but premature (line 6) as picking and transcribing a song does not achieve the intended project scope goals of the activity. Parker attempts to refocus the group with a compromise between the computational model and the application domain: relating controlling the sensors to singing (line 7). This compromise overloads ownership at performance runtime in both the computational model (who is holding and pressing the button that triggers the event which is a code entry point) and the application domain (for whose musical purpose is the computation running, like supporting the owner's singing). Bodhi recognizes her duty as a youth leader to model expected participant behavior and contributes to including the computational model in the conversation by relating the application domain object (O card) *melody* with

the computational model (specifically, the distributed system portion), e.g. she physically gestures to the BTU to reaffirm its presence with the group (lines 8-9). Carson voices the peer response, which went unchallenged after this excerpt, that some factor of concretely scoping the design of the project invoked a belief that she or the group might encounter difficulty completing the project (unclear if she was referring to the program implementation or the live performance aspect as *scary*) (line 10). In considering performance (accomplishment) outcomes, the participants in this excerpt encountered negative self-belief when committing to the ideas that they exercised their agency to create. From an instructor perspective, however, the biggest threat to successfully completing a project was believed to be the default group behavior between committing ideas on the poster: drifting back to transcription of long songs without limiting scope.

```
1 Bodhi (yl): The left hand can be two different synths so,
2             say Delaney controls the synth and Avery controls the other synth,
3             and its like, control the volume, or something, like pitch.
```

Bodhi explains her understanding of the computational model for solving an application domain problem. The problem they encounter is that, to implement their stories, two users would have to affect the same musical program. In the music they are transcribing, they look at a YouTube video with piano tablature which has a *right hand part* representing the melody and a *left hand part* representing the harmony. Left hand and right hand are musical terms describing which hand would be used for that music if played on a physical piano instrument. Bodhi suggests using the distributed system concept from the computational model to solve the problem, where they would distribute the music across two synthesizers (line 1) and give ownership of each synthesizer to a person (line 2), and then implement the user story as two independent programs (line 3). This solution fits the ownership model they had devised and demonstrated knowledge that the performance was made up of several running programs. However, their problem could also have been solved by using multiple message handling entry points in a single program on one synthesizer. Their solution would require implementing the transcription of the left hand part's music across two synthesizers' programs, respectively, rather than just one which could potentially double the implementation effort. The participants could work around this effort by using BlockyTalky's code importing and exporting IDE features, but these features were not taught to the participants as part of expected use and were intended to only be used by instructors encountering unexpected technical issues. So, how the learners expected to actually share the implementation across two synthesizers was unclear and doing so did not readily fit the computational model instructors showed BlockyTalky having and would have to be coded redundantly by hand. Potential adverse effects this solution would have on scope was not noticed or discussed by participants.

```
1 Bodhi (yl): This is hard, right?
2 Delaney (yp): [pointing to design poster] but I control...?
3 Bodhi (yl): We changed it
4             [the use of it refers to the left-hand and
5             right-hand piano transcriptions of the song]
6 Delaney (yp): So, what's my job?
```

Youth leader Anegla pair programs with youth participant Delaney to implement the left-hand part. Youth participants Carson and Avery move away to pair program the right-hand part. They move away since they believe they need to listen to the song from the YouTube video with piano tabs as they are transcribing the notes into pitch and duration format for coding into BlockyTalky. Being too close together would cause audio interference if both videos were played at the same time (making transcription difficult). They were unable to use headphones since they both wanted to hear the video. In beginning this translation, Bodhi comments on the difficulty of the project (line 1). Delaney, who has not participated much in the camp until this point, refers to their user stories written on the poster to clarify confusion about why they were transcribing the left-hand part rather than the right-hand part (line 2). She uses control to ask a question about ownership during the performance runtime (like what sensor she will be holding). Bodhi addresses the confusion by implying that their pair and the other pair had switched left and right hand part ownership (lines 3-5). This ownership changes both the runtime ownership and the implementation responsibility. Delaney, more confused, clarifies further asking what her job is to participate. The use of the word job contradicts the activity's student agency goals of wanting participants to have some autonomy over their aspect of the performance while still collaborating on a single performance in groups. Her role in the team being presented as a job implies that decisions would be made for her and then she would perform them.

```
1 [sharing out to all participants as an end of activity summary]
2 Avery (yp): [reading from poster] Delaney will be doing the controls of one synth,
3           and I will be doing another synth,
4           and Carson will be singing the song,
5           and then Bodhi will control the volume with ultrasonic sensor.
```

Avery who had been acting as the scribe for the group during the previous parts shares the results with the participants as a whole when the instructors asked the groups to reconvene. She relies on the poster depicted in Figure 4.2 and reads a summarized version of it. Avery uses the phrases *doing the controls* and *doing* as synonyms for the ownership verb *control* as was used in the group work consistently (lines 2-3), but omits the application domain detail of the left and right hand parts as presented in the poster. The other groups used different musical transcription techniques and may not know that left and right hand was the jargon this group decided to use for melody and harmony which was used by the instructors. She reads Carson's user story; it is purely in the application domain and could not have been constructed syntactically from the cards (line 4). She reads youth leader Bodhi's story as written on the poster verbatim which includes the user, verb, attribute, and sensor card contents, but this user story omits the object card content, so it is unclear whether the volume of the right hand or the left hand will be controlled. By using the overloaded verb controls as ownership on the attribute instead of the object, the implementation detail of how the volume would change with respect to the ultrasonic sensor was omitted. In the accompanying poster, the visual representation of the performance was purely in the application domain and recreated album art of the song they chose to perform including its name, in both English and Chinese, and its composer's name.

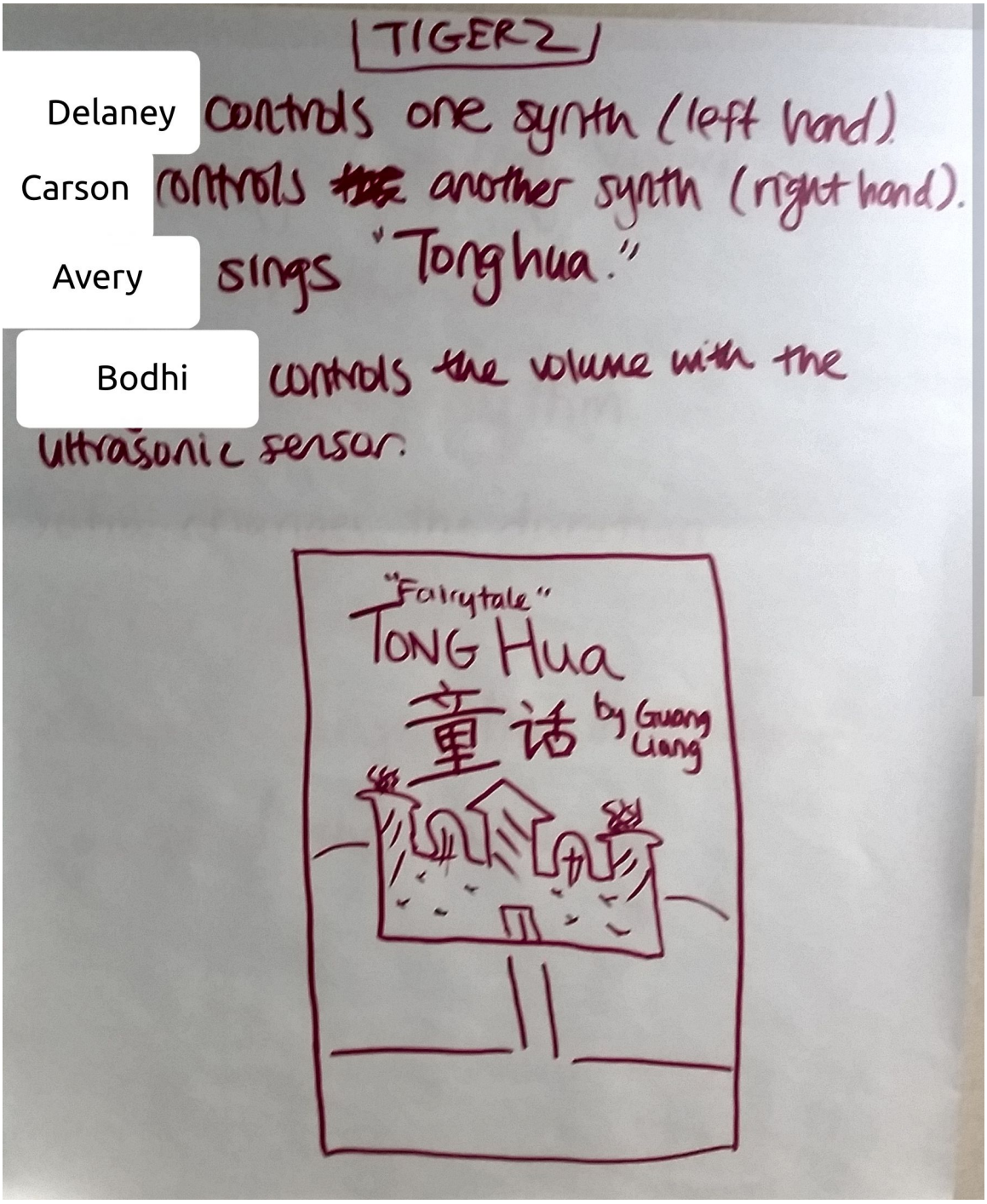


Figure 4.2: The finished user story poster for Bodhi, Carson, Delaney, and Avery with their four user stories and an illustration conveying the content of their performance.

In addition to these transcribed observations, undergraduate instructors presented the following field notes on this group.

```
Group one had been distracted at first,
flipping through Carson's binder of band songs
(I was excited that she brought it though!).
I suggested they think of what they would do
for their final presentation/performance
and they reined it in right away
(put the binder away, closed the laptop with YouTube).
They seemed nervous about doing a final presentation. . .
maybe there's something else I should be calling it.
Carson wanted to sing, but didn't really want to do it alone.
I think they're planning for her to sing solo for now,
but perhaps we can coax Avery or even Sofia into joining her.
Bodhi definitely does not want to sing.
```

In these notes, the preoccupation with non-committal progress in the application domain was viewed both negatively and positively towards implementation goals. In the negative, exploring the application domain separate from BlockyTalky distracted from scoping the project or thinking in terms of the computational model. In the positive, students had agency to use existing knowledge of the application domain as represented in third-party tools like the binder of music to relate the work to an aspect outside of the course. Introducing the idea of finishing a project and performing it correlated with nervousness which we view as a detriment to self-efficacy in how participants perceived their proficiency. However, knowledge the group demonstrated through vicarious experiences was viewed positively as the abilities of group members determined and reflected on their respective roles in the application domain like who would sing, who would not, and why.

```
Carson offered to write the code.
It is worth to note
that she is held in high esteem by her peers
and while she does not have a dominant personality,
she is the one who leads the group
```

In these notes, the belief that a particular participant acted in a way consistent with high self-efficacy and leadership was made readily apparent by the behavior of the group members. High self-efficacy seemed to correspond with a desire or self-perceived ability to implement the code for the project as well as lead.

```
While they have thought of the song
they have probably not thought of how to code it
Weyin and Avery said "This class is exciting"
```

```
I think in retrospect, its hard for them to know what they want to be in the system.
In the video log at the end, Avery said that performing the song was the most exciting part.
```

In these notes, the restriction of the workshop to an application domain of music was motivating to students despite perceived difficulty in learning multiple concepts at once: the application domain of musical performance and the computational model of distributed and message passing computing with sensor hardware. This analysis supports that student agency in the application domain encourages self-efficacy through mentions of excitement, but may not be perceived as beneficial to technical proficiency in the computational model.

4.2.4.2 Group three with youth leader Owen and youth participants Kelsey, Marley, and Lane

```
1 Kelsey (yp): [Puts her cards in order U,S,V,A,O]
2 Owen (yl): [inaudibly suggests order U,V,S,A,O]
3 Kelsey (yp): [to Marley] So, what is yours?
4 Owen (yl): [interjecting] She doesn't even know what she's doing yet.
5 Marley (yp): Marley generate ultrasonic pitch melody.
6 Owen (yl): Owen touches the button to increase the volume.
7 Lane (yp): Lane touches the toggle to change the duration of the harmony.
```

Youth participant Kelsey organizes her cards (line 1), but youth leader Owen suggests an ordering that she thinks will make it easier to complete the task (line 2). Kelsey asks youth participant Marley to share, and whether Kelsey is asking her to share her sentence or her cards is unclear (line 3). Since Marley has participated the least in this group until this session's activity, Owen responds in an attempt to relieve the expectations on Marley (line 4), but it is unclear how this affects Marley's perception of her expected participation in the group culture. Marley answers Kelsey's query by reading her cards in U V S A O order (line 5) which was Owen's suggested order. Without confirming or providing feedback, Owen reads her own cards as a grammatically correct user story in U V S A O order. Lane follows the example and reads her cards as a grammatically correct user story in what would appear to be U V S A O order, like Owen's. However, on the cards Touch is a sensor (S card), not a verb (V card) as the touch sensor is a button. Owen's user story omits an object (O card) from the application domain of music and Lane's user story interprets toggle as a sensor and not a verb. For example, an expected user story using Lane's cards could be "Lane toggles the duration of the harmony using the touch sensor". Toggle, as a verb, was unclear to students and had to be explained by instructors as a jargon term meaning to switch on-to-off or off-to-on depending on a current value. A domain expert might elaborate Lane's user story as toggling between two states for some duration parameter.

```
1 Kelsey (yp) Kelsey.. oh.
2 Owen (yl): How are you supposed to use light?
3 Kelsey (yp) Kelsey put on the light.
```

```
4 Group: [Laughs at the tone of the response]
5 Lane (yp) + Owen (yl): That makes sense!
```

Kelsey goes to share her user story, but encounters confusion (line 1) which she clarifies before simply reading her cards in a grammatically correct form. Owen anticipates Kelsey's confusion and asks the question she believes Kelsey has (line 2). Kelsey answers the question with a tone of voice that implies the answer is not serious with a humorous overtone (line 3) and the group responds to that intention by laughing with her (line 4). This interaction between Kelsey and the group members appears to be not mean-spirited and expresses humor at the perceived lack of knowledge. Kelsey's answer shows that she interprets the S card labeled light as a source of light rather than a sensor that senses light (intensity or color). However, Lane and Owen immediately affirm the answer (line 5) since her user story makes syntactic sense (it is grammatically correct). From interactions like this, grammatical correctness appears to be the implicit group consensus of the acceptance criteria for the activity.

```
1 Marley (yp): [reorders her cards to Marley generates pitch ultrasonic melody]
2 Group: [motioning, asking if Marley needs to change a card]
3 Marley (yp): nah
4 Owen (yl): It makes sense, you [Marley] are just wording it differently.
5 Kelsey (yp) [assisting Marley] Marley generates the pitch of the melody by using ultrasonic.
```

Marley struggles with mimicing the group so far by converting her cards into a grammatically correct sentence despite having shared her card contents first. She attempts to fix her issue by reordering the cards implying that she believes her issue in converting the cards into a sentence depends on their order (line 1). The group attempts to diagnose the cause of Marley's confusion, initially guessing that she understands the instructions, but does not like the content of the cards she is given (line 2). Marley denies their help, but it is unclear if that means the group's assumption was wrong (line 3). Owen reassures Marley by asserting her story's meaning is correct and her efforts are valuable, despite some perceived issue in her presentation of the knowledge (line 4). Kelsey attempts to bridge this gap by converting Marley's cards into a grammatically correct sentence (line 5). In her tone, Kelsey expresses her confusion at the semantics of the sentence, specifically "generating a pitch". To an instructor observer, Marley's confusion most likely stemmed from this semantic error where the resulting sentence made logical sense in neither the application domain nor the computational model which caused an inhibition to participating. Knowledge was constructed by the group only by debugging this error verbally and a consensus of acceptance criteria began to include semantics as a component.

```
1 Kelsey (yp) Kelsey turned the light on and changed the melody of the instrument sound.
2 Ben (up): Can you give an example of what that might be like in a performance?
3 Kelsey (yp) No [immediate answer, upward vocal inflection, honest]
4 Ben (up): No? Ok. So the light sensor controls the melody. And you're doing that?
5      No? You look confused.
6 Kelsey (yp) I am.
```

```
Ben (up): [waiting for visual affirmation between each question]
     It means, Kelsey uses the light sensor to change the melody.
     And the way she changes it is changing the instrument sounds.
     So you know how there were different kinds of instruments you used last time?
     Before they all sounded the same and now they sound different than each other?
     So this would be you use the light sensor somehow
     to change what it sounds like in the melody.
```

In sharing their poster with the group as a whole, Kelsey reads her sentence as written (line 1). Recall that the group affirmed Kelsey's sentence, answered questions, and felt satisfied with the answer once it became syntactically correct and semantically reasonable. Ben asks a follow up question in an attempt to see what implementation Kelsey would write to satisfy that user story (line 2). That is, the user story describes *what* the code will do in the computational model and musically, but not *how* the code will do so, specifically. Kelsey immediately gives an honest negative answer, but does not attempt to further think about or answer the question (line 3). It is interesting as there was no intention to correct Kelsey, and she previously had confidence in her answer by building an understanding in her group. But, in this larger group setting, she responds as if she believes her answer is wrong (lines 5-6) rather than the intention of collaborating on a more complete scenario (lines 8-13). The setting of this excerpt, sharing with the whole class, changed the participant's construction of knowledge. The rapport observed in soliciting explanatory feedback in previous excerpts came through refining understanding amongst peers. In this interaction, an assumed culture of knowledge as given from an instructor to the participant group as a whole interfered with helping a particular participant construct knowledge with the instructor as a collaborator.

In Figure 4.3, the user stories are listed at the top, with Kelsey's user story purely in the application domain without any computational component. Marley's sentence encompasses two technical user stories, one using the ultrasonic sensor to control piano volume and one using a touch sensor to toggle whether the piano instrument is playing or not. It is unclear what musical role the piano instrument will be playing, like melody or harmony, but from the other stories it seems like the piano is intended as harmony accompaniment. Lane's user story follows from her cards and was ammended to not omit the sensor she would use. Owen's story is similar to Kelsey's story where her role is completely in the application domain with no computational component. Their flyer art uses names to symbolize the performers underscored by their musical roles and the name of the song they will be performing. The flashlight icon has no computational component and is put there to symbolize the name of the song which is "Flashlight". Post-activity feedback from other participants notes that ideas this group could pursue include using different kinds of sensors as well as general encouragement. Small sticky notes were used by the group to indicate progress so they could remember their place for the next session including that they had the first 40 notes encoded into their BTU, that they had left off in the sheet music on a note with pitch F4 which corresponded to 21 seconds in the YouTube video they were transcribing from.

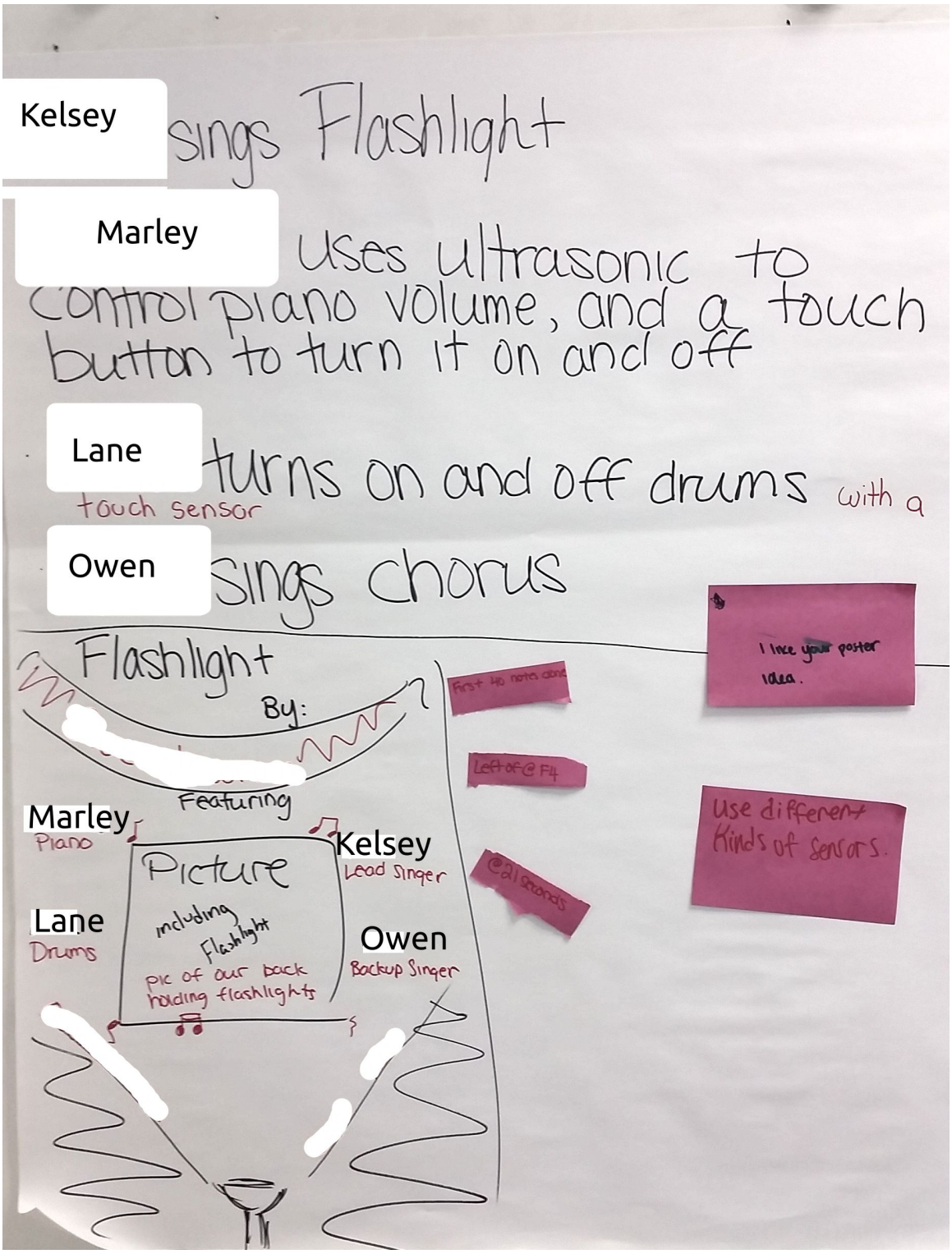


Figure 4.3: The poster containing the user stories and promotional flyer art of the group with youth leader Owen and youth participants Kelsey, Marley, and Lane

.

4.2.4.3 Group two with youth leader Jude, and youth participants Harper, Finley, and Ellison

Group: [watching video of song they want to do]
Matt (gi): So here's my question, what are the parts of all of this [song]
that you are going to map to these [holds up cards].
Ellison (yp): I'm going sing parts of it.
Matt (gi): So that's the melody. So you're going sing parts of it right?
Are you going to have it play the melody with you?
Finley (yp): What's the melody?
Ellison (yp): Probably this song! [laughs]
Matt (gi): So, what are you going to do to get it to support or react to you while you sing?
Ellison (yp): What do you mean?
Matt (gi): So you're singing right? So is it going to play the melody with you to help you sing?
Ellison (yp): Yeah.
Matt (gi): But you know, these kinds of songs, they emphasize stuff by harmonizing.
So how are you going to get it to harmonize with you?
Ellison (yp): [looks to group] I don't know..
Matt (gi): You have it in front of you, you can use those [cards] to help you.
Ellison [Picks up card] Pitch!
Matt (gi): Yeah, you can use the ultrasonic to decrease the pitch to harmonize with you.
Ellison (yp): [thinking]
Matt (gi): So you're singing, and you can get closer to it.
Jude (yl): And it can make the beats of the song too, right?
Matt (gi): So she is singing, and it plays the melody the first time with you so you can hear the pitches.
Group: [confirming]
Matt (gi): Then do something, while you're singing, to make it change pitch to harmonize with you?
Ellison (yp): [nods in affirmation, but then interjects]
Wait what? I don't understand anything because
I don't know what you're talking about.
Matt (gi): Do you want a computer [referring to a laptop] in front of you while you're singing?
Ellison (yp): Yes
Matt (gi): So, you're standing up and you're singing,
Do you want to be coding while you're singing?
Ellison (yp): [reluctantly] No
Matt (gi): That sounds messy right?
Ellison (yp): mhmm.
Matt (gi): So, you have this ultrasonic, it knows how far away you are from it.
You're singing. You get one time through the opening melody.
And it comes time for the chorus and you want an impact.
What do you want it to do?
Ellison (yp): Decrease the pitch.

```
Matt (gi): How are you going to do that with the ultrasonic?
Ellison (yp): [no response]
Matt (gi): Ultrasonic knows how far away you are from it, what are you going to do?
Ellison (yp): Get closer
Matt (gi): So you are going to get closer to decrease the pitch? There's one!
Group: [visible relief]
Ellison (yp): So I'll use the ultrasonic? Yeah.
Ellison (yp): [to group] So then I go like closer [motions] it decreases the pitch.
Jude (yl): [Drawing stick figure pictures on the poster to
        illustrate the physical part of the story]
```

In this excerpt, graduate instructor Matt attempts to answer a "why are we doing this activity" question from the group that is preventing them from beginning the activity. Until approached to answer this question, the group was doing a mix of putting the cards in order as well as watching a YouTube video with the song they had chosen to perform. They were not discussing the video, not writing down information from the video, and not writing user stories (line 1). It was unclear to Matt what was hindering their progress, so he attempts to address the confusion by walking through building one of their user stories together, collaboratively. In this interaction, the youth participants remained in good spirits. However they grew anxious that they weren't understanding the answer when each of their responses was answered with further questions rather than affirmation. Matt and Ellison framed the problem by moving from what song they were planning on performing (which the group was fairly confident in choosing) to how they were going to perform it and implement it (line 1-12). Questions from youth participants often were posed in a joking manner, without self-censorship, and in terms of the application domain when they intended to solicit responses from their peers (lines 7, 21, 25-27). However, questions posed for clarification to Matt were more reluctant and were difficult to move from yes-and-no style (lines 12, 29, 34) to content exploration in the computational model (lines 17, 39, 46-47). As epistemological messaging, Matt's used iterative questions to direct student attention from the content for several pieces of the computational model and application domain (singing, the ultrasonic, and volume) to the production of the user story (going closer affects the volume), but does not model the form given in the activity. However, Ellison's response added missing pieces that fit the form (the ultrasonic sensor) and Jude writes it in the grammatically expected form. This modeling provided structure for the activity in the moment, but the provided examples accidentally became corrective feedback (instead of one of many choices) and reduced student agency to which student designs likely overfit. Specifically, Ellison's cards could be interpreted in many user stories that would work with the song they chose, but by walking through a technical example this way, the group added it to their poster uncritically, and it is unclear if that was of their own desire or not.

```
Finley (yp): [organized cards]
       I press a button and then synth 2 plays
       and then, uh, to play a synth.
```

```
       and then when I release a button, it stops playing.
Jude (yl): What, when you release?
Finley (yp): When I release the button, the synth stops.
```

Youth participant Finley deviates from her cards syntactically, but forms a complete user story with two events purely in the computational model (lines 1-4) and omits the component from the musical domain (i.e. what musical concept the synth is performing). Youth leader Jude writes everyone's stories on their shared poster when asking for the clarification (line 5). Finley confirms the story without changing or omitting major pieces.

```
Harper (yp): [sarcastic] Can I bring a bed into the performance
Matt (gi): If you make it an instrument, sure.
...
Jude (yl): [to Harper] What are you going to do
Harper (yp): [shrugs]
Jude (yl): I'm going to draw on your hair and make your hair red
Harper (yp): I'm going to sleep
Matt (gi): And what are you going to do when you sleep.
Jude (yl): Maybe if you sleep, the rhythm changes.
       When you close your eyes, the light..
Rory (ui): yeah the light!
Matt (gi): I see you constantly put your head on the table.
Harper (yp): Oh yeah
Matt (gi): So when your head goes to the table,
      it covers the light sensor, so your sleeping could be an instrument.
      What do you want it to do?
Harper (yp): I don't know.
Matt (gi): So, when you fall asleep, and your head hits the table, what happens to the rhythm.
Harper (yp): [shrugs]
Matt (gi): The rhythm is usually done with two sounds, hard and soft
      [referring to two settings in the instrument choice drop down menu],
      so what happens to the drum sounds.
Harper (yp): It gets, lower?
Matt (gi): It gets lower? Would you want it to wake you up or put you to sleep?
Harper (yp): Not wake me up.
Jude (yl): So when she puts her head down, it gets softer.
```

Youth participant Harper was the least engaged participant in this group and in the camp overall. However, this activity was the first that became able to include her contributions. Harper tries to excuse herself from responsibility by dismissing requests to participate but is denied with the same level of levity (lines 1-2). Youth leader Jude attempts to include her again (lines 4-5), and relates her joking to a part of the computational model (line 6) as she motions to a sensor that can detect colors

to imply she can get Harper to passively participate. From the tone of group conversations in the video, there does not appear to be animosity or assumptions of low proficiency between Harper and other participants, but everyone expected her non-participation from her prior behavior. The rest of this excerpt is not strictly analyzed to avoid overemphasising the negative proficiency and performance aspects of the youth participant from this observation. By analyzing this excerpt, we aim to acknowledge the criticism of student agency teaching interventions at introductory levels from lack of structure. As a note of interest in building understanding, Harper introduces the word *lower* (line 23) to have semantic meaning synonymous with softer in timbre or volume from the application domain (not to be confused with lower in pitch). Instructor Matt confirms her concept of lower by asking her to clarify her association concretely with the shared assumption that softer sounds would not wake her up (lines 24-25). Since she had ownership of the drum role, lower in terms of timbre could mean switching to a drum of lower pitch (like a bass drum) which would change Matt's model of her use of the term lower. In clarifying, Jude affirms the collected knowledge by using the synonym *softer* as a result of an event in the physical space. In terms of computational thought, Jude's affirmation connects the physical event input to the physical actuation output and abstracts over the complete computational system (line 26).

It is worth noting that, in this activity, youth participant Harper contributed more than she had been in previous workshop sessions by using this design activity as a starting point. So, we include this excerpt as context of our belief that she had a low self-efficacy expressed as non-committing and a low desire to exercise her agency until we re-framed the project in terms of the application domain..

From the undergraduate instructors field notes on an overall experience with Harper:

```
Harper had the sheet music for “See You Again”
and asked me to dictate the notes to her.
I read the notes and chords for the intro and chorus,
and she wrote them down on a sticky note.
I did not say what notes were in which octave
and for how long they were to be held for
(the sticky note just read C Bb C A, etc.),
so I hope they are able to experiment with the beats
and the play blocks to figure out the correct sound
without spending too much time on the transcription part.
```

In these field notes, an activity in the application domain was seen as doable and interesting by the previously unengaged youth participant Harper, and the instructor constructed the next steps for success and difficulty with relation to engaging with BlockyTalky.

From another instructor's field notes:

```
I managed to convince Harper
to do small parts of the music transcription.
First, Parker interpreted and read the sheet music
```

```
while Harper wrote down the notes on a sticky note,
and then I read the notes to Harper as she made the code.
(She did all of the code!! :D)

I should have had Jude read the notes back to Harper (oops :( ).
But Harper was being much more motivated than I've seen her recently,
even though it was a really boring and tedious process.
I did show her ctrl+c and ctrl+v.
```

In these field notes, a technical process was developed between the undergraduate instructors and the youth participant to make progress. The computational thinking applied was not in the domain of the computational model of the tool (distributed systems, event based computation, or sensors). The computational thinking Harper expressed was in the translation of one data representation to another (an unintended PL concept).

```
1 Rory (ui): Is yours the one with the light sensor?
2 Jude (yl): I don't know because its almost the same as hers with the ultrasonic
3 Rory (ui): That's ok. You can use two ultrasonics.
4 Jude (yl): Oh yeah, we can have multiple ultrasonics.
5        [writing it out] ..generates the volume of the drum?
6 Rory (ui): You can say that, but if you don't like the word generate you can change it a bit.
7        Like controls the volume of the drum.
```

Undergraduate instructor Rory notices that youth leader Jude has written every user story except her own on the poster (line 1) and offers help from the prospective of the computational model. Jude poses that her problem is that her user story is somehow similar to Ellison's and it is unclear whether her problem is a question of the application domain or the computational model (line 2). Rory responds assuming Jude does not understand if the computational model allows a single program to have two of the same sensor (line 3). BlockyTalky does allow two of the same sensor since input is limited by physical port and non sensor uniqueness. Jude confirms that the answer helped her since they actually possess multiple ultrasonic sensors in addition to knowing they can potentially use more than one at a time (line 4). As Jude is writing it out, she vocalizes a part of the story that, while syntactically correct, seems funny semantically (line 5). Rory responds to the potential question saying that Jude has agency over the verb and can choose a different one, and uses the verb controls as an example.

```
1 Rory (ui): [Describe] what your project is.
2 Finley (yp): We're doing ``see you again''
3 Ellison (yp): I'm singing, and then I'm using the ultrasonic,
4        and when I go closer to it, it increases the pitch.
5 Rory (ui): The pitch of what?
6 Ellison (yp): the song.
7 Finley (yp): I press a button, and then a synth plays,
```

```
        and when I let go, it stops.
Harper (yp): I put my head down and control it.
Rory (ui): What does it control?
Harper (yp): The drum.
Jude (yl): I'm using the ultra sonic to change
        the volume of the drum or piano.
```

At the end of the session, the group was asked to share with Instructor Rory who was actively operating one of the cameras to record a video log-style summary of the group's work. Rory tries to ask follow up questions on ambiguous uses of the word *it* which hide the conceptual understanding of the project. Both Ellison and Jude present their user stories in the form of "I am using the sensor" and include details on what attribute of the performance will be changed. Finley's user story remains unchanged from her initial draft and still does not include the musical concept the synth will be playing (e.g. melody, harmony, etc).

```
Rory (ui): What are you most excited about?
Ellison (yp): What do you mean?
Rory (ui): [Gestures for the to attempt to answer the prompt without clarifying information]
Ellison (yp): Singing
Jude (yl): Putting it all together.
Rory (ui): What might be the hardest part?
Ellison (yp): the notes, finding the notes?
```

In this except from the end of video log, undergraduate instructor Rory asks questions to gauge how students felt going forward with the project (line 1). At this point in the activity, several participants expressed worry at the shift from previous activities into working on a "real" project similar to group one. Youth participant Ellison expresses excitement from the application domain that we see common to many of the youth participants in the previous excerpts (line 4). This excitement was exceptionally valuable in encouraging participants who had not yet contributed to their groups in previous exercises. Youth leader Jude expressed that she was most excited for accomplishing a finished project (line 5). We found this sentiment common to the youth leaders as they had both coordination responsibilities as well as modeling what they thought expected, good participation should look like. We interpret the youth leader's sentiments as a belief that their work was successful if their facilitation lead to a finished, delivered project and performance. Rory asks what the most difficult part might be with the intention of getting the group to plan ahead and not procrastinate on programming until it is too late (line 6). Instructors assume the most difficult part of the project will be implementation details. Ellison answers that the act of finding the music in the wild, using online YouTube videos as a primary source, and encoding them into BlockyTalky might be the most difficult (line 7). Of interest in considering Ellison's answer, all groups had already made progress on transcribing YouTube or sheet music notation into BlockyTalky by the end of the workshop session. No progress had yet been made on coding any dynamic runtime behavior (like changing volume) using sensors from the user stories. Groups

were testing their musical programs at this stage by having the music play when the program was run at a initialization entry point rather than when sent a message due to some event entry point from a LEGO hardware BTU.

In Figure 4.4, the group's design poster contains the user stories which are more embellished grammatically and introduce ideas not present in the cards. Ellison's user story includes application domain information of what song is being sung as well as how the code will change the music by her getting closer. Finley's story remains the same from the discussion, but expands on the computational thinking of what the touch sensor can do. This expansion likely formed from an overheard explanation to group one of what the verb *toggle* meant, but with the change in state being physically embodied in the button's state and not logically stored in a variable. Harper's story was written including the physical action that would activate a sensor, but not the (light) sensor itself. The placeholder for what musical concept her action affects was filled in with the drum instruments, but was not added as an afterthought like the note about what song Ellison is singing. Jude's user story matched the syntactically valid form of her cards that was originally discussed, but with the qualifier that she could use a different instrument after she asked her clarifying question about whether her story and the other stories were too similar. In Harper's story, the drum getting softer from her action implies choosing a softer drum instrument sound (changing the timbre), but there was confusion if that would overlap musically with changing the volume in Jude's story. The poster gives context for Jude's earlier question about similarity actually being about semantic similarity in the application domain and not technical similarity in the computational model. At the bottom, Jude had illustrated each participant performing with stick figures, sensors and BTUs using boxes, and connections using wires and laptops. From an undergraduate instructor's field notes, they question how the participants think about the BTU's computational model and its dependence on other devices:

```
Group two took a less artsy/very literal approach to their poster,
and drew the four people in the group with their Blocky Talkies.
They (Jude?) drew the sensors connected to the BTUs,
and the BTUs connected to the laptops.

They're associating the BTUs with the laptops.
[Another ui] mentioned something about this after the last vlogs.
What's especially interesting is that they were
illustrating the performance, but still included the laptops,
so they think that the laptops have to be there to run the BTUs maybe.
(Although I guess literally speaking, you do need to hit “Run” on the laptop...)
Also, I wonder if they're just not paying attention
to where the power cord is going, and are thinking it's going to the laptop maybe.
```

In this observation, the role of the laptop in the runtime of the project becomes unclear and seems to have an impact on the participants reasoning about distributed computing from the computational model. BlockyTalky had a design goal of relating to first-time programmers by

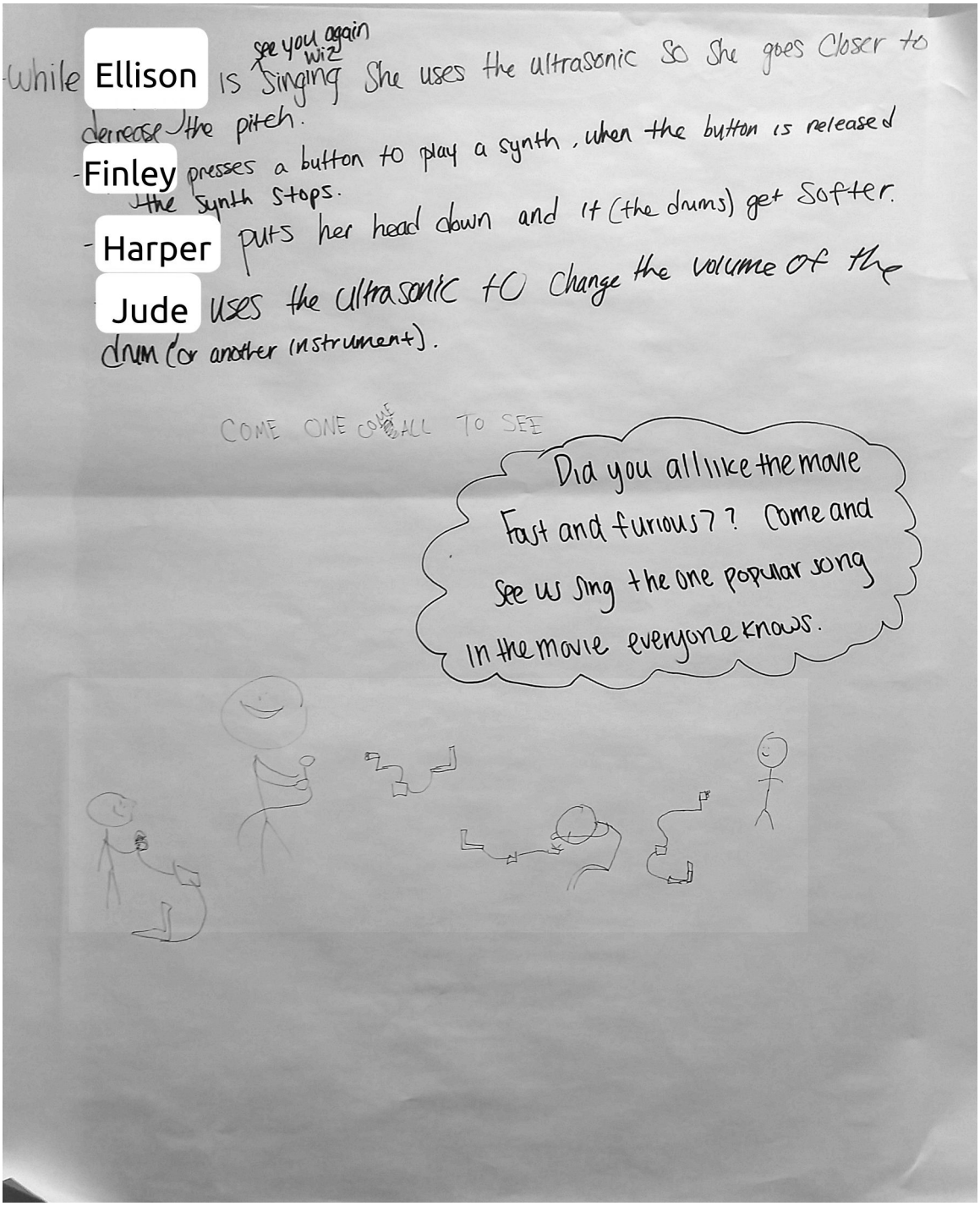


Figure 4.4: The end of activity poster with user stories from youth leader Jude and youth participants Harper, Ellison, and Finley.

facilitating the creation of programs like the distributed computing devices participants use in daily life, e.g. webapps, phones, and smart appliances. The incorporation of the webapp IDE as a necessary component to the runtime of the BTU (running the program) interrupted this design goal and potentially manifested in Matt's question to Ellison about if she would (or should) be able to change blocks while performing in the earlier excerpt.

4.2.5 User story activity discussion

We analyze these excerpts and field notes to construct a theory of how students thought about project designs in terms of the application domain of musical performances, the computational model of event-driven programming and message passing provided by BlockyTalky, as well as the relationship between the two. We selected excerpts that highlight the participant's agency in design decisions, self-belief in proficiency, and collaborative construction and affirmation of knowledge through vicarious experiences. With a DCOG approach, we elaborated on how understanding appears in the rapport and culture participants create between themselves, the tools, their peers, the youth leaders, and the instructors.

In the cases with participants who had not participated prior to this teaching intervention, they often contributed in terms of the application domain instead of directly in terms of programming in BlockyTalky which gave a non-technical entry into affirming computational thinking. Often, these participants had difficulty interacting with their peers, when previous interaction was mostly technical learning of the BlockyTalky language features. A lack (or perceived) lack of understanding of the technical tasks often made these participants wait to ask questions or not know how to ask questions. The peers often tried to help, but encountered epistemological messaging difficulties when having to query or guess the misunderstanding from the participant which may have fostered a self-belief of deficiencies in their non-participating peer. The same misunderstandings were taken as endearment when presented by a participant asserting themselves to their peers in an us-against-the-lack-of-knowledge framing with some levity. We saw how increasing levels of peership could negatively affect the feedback of constructing knowledge as the type of responses the participant anticipated gradually moved from explanatory to corrective. Specifically when interacting with other youth participants, high self-efficacy youth participants presented their misunderstandings as a way to explore the design space which avoided overfitting examples. Then, with youth leaders, they added slightly more expectations of a correct form of answer. With instructors, they overfit to modeled answers, but still tried to understand the meaning behind the feedback. With the group as a whole, they presented their knowledge as correct or incorrect and had the most difficulty elaborating or exploring the design space.

By approaching participation from the application domain, participants could contribute using plain language in the shared culture with respect to music (from a project design perspective, but not necessarily a technical perspective). In Harper's case, she had prior knowledge of music which made her group value her contributions and lead to her attempting the coding task participants perceived as most difficult first: translating music from notation into BlockyTalky code by an ad-hoc data transformation methodology using videos, sticky notes, an undergraduate instructor, and the IDE.

When looking at how groups approached the activity, the most interesting observation of self-efficacy was how they started and when they felt they were done. When participants asked for clarification of the activity instructions from instructors, they became more confused with open-ended answers modeling examples of what they *could* do. There was an implicit expectation among all groups that there were correct answers to the activity. There was no correct answer of the activity intended by instructors, instructors were looking for correct behavior (engagement and participation). In this absence, participants among all three groups ultimately settled on the same initial implicit acceptance criteria: grammatical correctness. The excerpts highlight instances where confusion was alleviated when participants arrived at user story sentences that sounded correct in terms of music and ownership, regardless of how their groups would implement them in the computational model. The benefit of this approach was participants determined ownership quickly; who would write what code with the user story sentences as reference. There were two major detriments. Participants thought they had done something wrong and expressed a lower self-belief of proficiency when asked to give examples of what their user stories might look like or when asked *how* their sentences would work. It was unclear where participants had trouble making progress (in terms of the activity, in terms of the application domain of musical performances, or in terms of the computational model) causing misunderstandings through overloading terms.

4.2.6 Design Documents

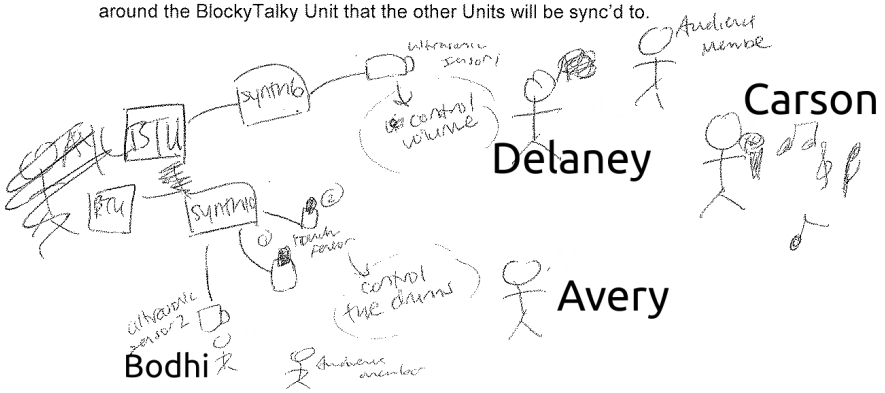
In the session after the user story activity, it was clear instructors needed to reinforce the computational model in order to help participants scope their musical performance. A design document teaching intervention helped participants outline the physical layout of their performance with the goal of informing participants of what sensors they might need. Then, participants would use the distributed computational model at a high level to associate a message with a musical action. If the participants had other ideas, like the end of one musical idea triggering another, they were encouraged to express it here using the message passing computational model. Then the document could be used as a checklist of everything participants would need to write code for (an event entry point in both the hardware and synth BTUs per item). Shifting away from generic ideas in the application domain would prepare their performances more tangibly by setting an initial project scope.

In the instructions, participants are asked to draw a red circle around the BTU that the other ones would be synced to. This instruction arose from participants asking how to coordinate multiple synthesizers for their performances since, in the initial implementation, the BTUs were not guaranteed to be on the same metronome at runtime. In response to their suggestion, we added a block and implementation for syncing one BTU's metronome to another so that if they both received a message to play at roughly the same time, their downbeats would occur within 40 milliseconds of each other (within reasonable human hearing latency for music synchronization). In reminding participants they had this new language feature, the instructions included it, however no participants included this circling in their design documents.

Figure 4.5 shows the design document for the group with youth leader Bodhi and youth

Technical overview

Draw a picture of what you will build. Show sensors, sensor units, and synths. Draw a red circle around the BlockyTalky Unit that the other Units will be sync'd to.



Document the messages that you will use to connect the pieces of your build together:

Message	Sender	Receiver	Action
get louder	H/A	Synth 6/10	person walking towards sensor
get softer	H/A	4/10	person walking away from sensor
play drums	H/A	6/10	clicking sensor

Figure 4.5: The design document to outline the message passing computational model for the project of youth leader Bodhi with youth participants Carson, Delaney, and Avery.

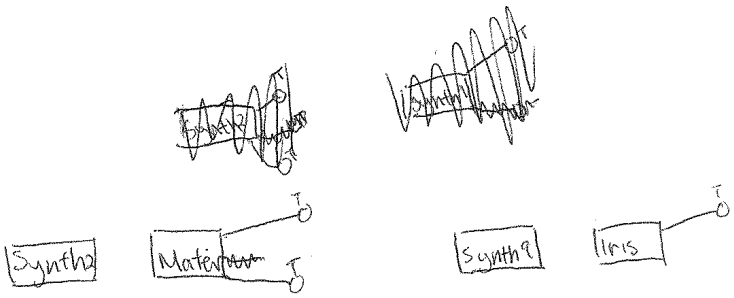
participants Carson, Delaney, and Avery. In their user stories, participants decided to divide responsibilities by ownership in the application domain: singing the melody, controlling the melody instrumental, and controlling the harmony instrumental. Youth leader Bodhi, understanding more of the instructors' goals in teaching the computational model, took ownership of changing an attribute of the performance, the volume, at runtime using the ultrasonic sensor. In their design document they had changed the computational goals since the user stories poster. They decided to start the piano accompaniment upon running the program. Then, during their performance, they would use the ultrasonic to change the volume of the accompaniment by having an audience member or group member (it was undecided at the time) walk towards or away from the sensor. Because they did not want the drum sounds to start immediately at the beginning with the piano, they decided to have it start when pressing a touch sensor. Boxes in the drawing represent hardware BTUs, synth BTUs, and sensors. Stick figures represent participants and audience members. Lines represent logical connection and are overloaded. Physical connections between the BTUs and the sensors are drawn with lines as well as logical send and receiver connections between the hardware BTUs and the synth BTUs. The hardware BTUs are labeled BTU instead of their individual names while the synths are given their individual names (synth and then a number). Sensors are connected with lines to the synth BTUs instead of the unnammed BTUs which we would assume would have been the hardware ones that the sensors should be connected to in practice. The three messages they outlined would implement this project. However, they interpreted the action column to mean what event would occur to trigger the message instead of what action would happen as a result of the receiver receiving the message which was the intent the instructors had for the activity.

Figure 4.6 illustrates group three's design document which accurately depicts the computation that would implement their project. The BTUs are present, and sensors are physically connected to the hardware BTUs. In the illustration, there is no visual representation of message communication. In wanting to perform a song's accompaniment with no variable behavior, the hardware BTUs became button controllers that signal to the synth BTUs to start the musical accompaniment the group would sing along with. The instruction for describing the action is interpreted as the action that results from receiving the message. The directness of their implementation appears in their overloading of the word *play* as the message string, and that the purpose is to start the piano accompaniment. Captured in this document is a directive that the drum will be played during the chorus. In actuality, when the drum will play is a runtime determination since it depends on when the play message is sent to the synth. From this document, we determine that there are aspects of the performance that the group wanted to capture that was not afforded by designing a message passing protocol in this representation.

Figure 4.7 illustrates group two's design document which uses the illustration to depict not only the physical arrangement of the BTUs and their sensors, but also the performers and their roles. The term action is interpreted as the event that will cause the sender BTU to send the message to the receiver BTU except the first entry also includes the effect on the performance and presents the sender as the sensor rather than a named BTU. The messages are chosen similarly to group three's as simply starting the parts of the song with the message string *play*, however

Technical overview

Draw a picture of what you will build. Show sensors, sensor units, and synths. Draw a red circle around the BlockyTalky Unit that the other Units will be sync'd to.



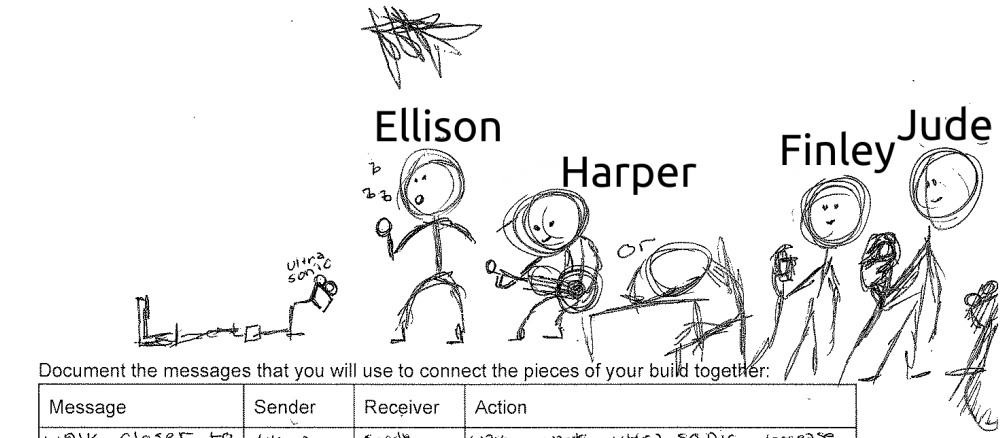
Document the messages that you will use to connect the pieces of your build together:

Message	Sender	Receiver	Action
Play	Meter	Synth2	plays piano
Play2	Iris, Meter	Synth9	plays piano
play	Iris	Synth9	plays drum during chorus

Figure 4.6: The design document to outline the message passing computational model for the project of youth leader Owen with youth participants Marley, Kelsey, and Lane.

Technical overview

Draw a picture of what you will build. Show sensors, sensor units, and synths. Draw a red circle around the BlockyTalky Unit that the other Units will be sync'd to.



Document the messages that you will use to connect the pieces of your build together:

Message	Sender	Receiver	Action
walk closer to sender to decrease pitch	ultra-sonic (controlled by Ellison)	synth	walk towards ultra sonic decrease pitch of song
play chorus	poseidon	synth 7	push button
play intro	zeus	synth 3	push button

Figure 4.7: The design document to outline the message passing computational model for the project of youth leader Jude with youth participants Harper, Finley, and Ellison.

they are annotated with specificity of which part is being played. In the first line, the message is overloaded with the condition that triggers the event that sends the message. From an instructor's perspective, this misunderstanding is preferable to under-specifying the message as it relates the performance to the computational model.

4.2.7 Design document summary

In these design documents, we see how giving student agency in the application domain affects the experience of exploring the computational model. The benefit is that performance aspects, ownership, and responsibility delegation are all captured in this design. The detriment is that these programs will be very simple with respect to the computational model, and most effort involves data transformation from the musical domain into blocks. Work since this study, using BlockyTalky, encourages shorter musical motifs coordinated in more complex protocols by restructuring the application domain to be algorithmic music design rather than performances [SKA+17].

4.2.8 Resulting code

To illustrate both the success and issues of the approach, figure 4.8 shows a sample of BlockyTalky code written by group three to handle the message "play" on BTU synth2. As a tinkering based approach, writing the code, changing the parameters, and testing it often was a very valuable computational thinking work loop for parameters and debugging. The group knew the project they wanted to implement, designed how it would work, and achieved it. The surprising factor was the value students found in programming a single musical idea, a motif, containing more than 60 sequential note blocks to perform a song. The biggest issue was instructors not realizing the participants' learning while facilitating the activity when group knowledge was being constructed in more subtle ways when they were discussing the musical performances and not actively coding or writing. In the moment, the agency students exercise looks like it inhibits understanding of the computational model and from the code artifact alone that might be the conclusion we would make. However, participants applied computational thinking to the application domain that would not have arose if we told them to simply implement a project we wanted to demonstrate all of BlockyTalky's language features and computational model.

4.2.9 BlockyTalky discussion

In this teaching intervention, we provided framing agency opportunities with respect to the goals of the workshop by letting students determine the acceptance criteria for the design activity and satisfying criteria for a musical performance project. Since not all song or performance choices require equal effort, the activity intended to help participants scope their projects by creating a user story sentence per participant. From these sentences, the participants were expected to define the minimal performance and coding responsibilities from the bottom up (as opposed to satisfying criteria from the top down, i.e. how close the performance is to the youtube video).

The activity itself provided a form, but left the structure open to editing at the participants' discretion. Groups exhibited epistemic agency for learning when collaboratively authoring

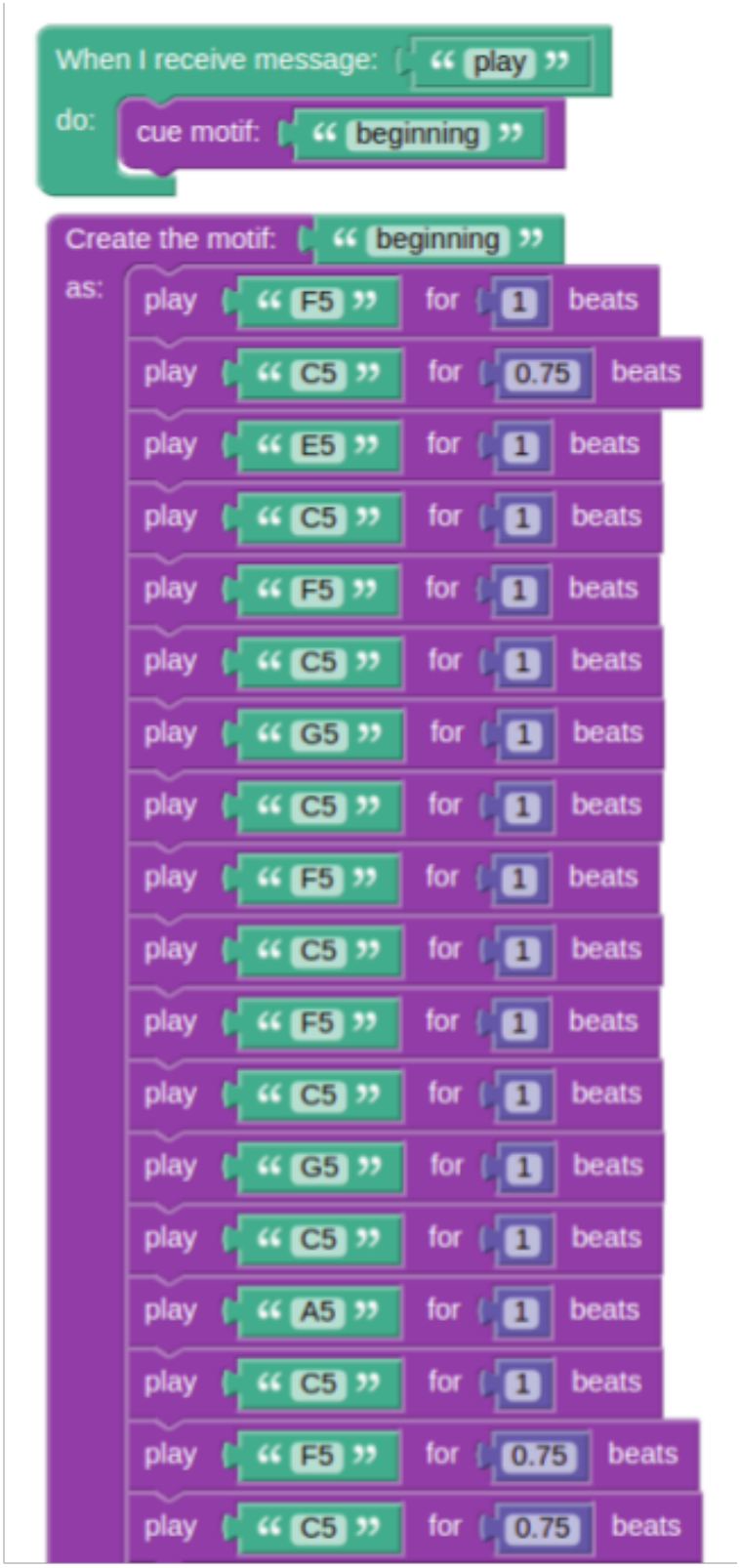


Figure 4.8: A sample of a synth program from group three where the motif extends for more than 60 more play note blocks.

the acceptance criteria for the activity. Groups exhibited epistemic agency for intrinsic value when putting their results on their design posters. Three primary acceptance criteria emerged: grammatical correctness, semantic correctness with respect to musical performances, and usefulness to the project. We observed the correctness acceptance criteria with discussing the sentences in the groups; participants were trying to discern the instructions of the activity and made choices about whether they had done the activity correctly. We observed the usefulness to the project acceptance criteria when the groups' put the sentences on their posters and chose forms that would have been impossible using the cards, i.e. Avery sings "Tonghua".

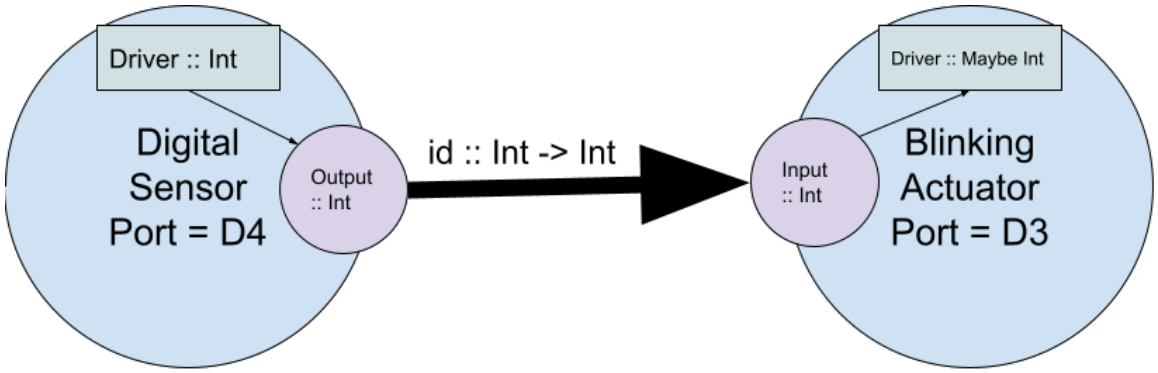
Participants showed positive self-efficacy while exhibiting the epistemic agency for learning. When discerning the correctness of their sentences, participants presented incomplete knowledge for explanatory feedback with their peers and the undergraduate instructors. Affirmation from their peers included reassurance that perceived *wrong* answers were actually mostly correct or could lead to the correct understanding of the group. Affirmations of semantic incorrectness with respect to musical performances enabled previously low-participating students to participate by exhibiting agency for practice, i.e. devising ways to encode the music into BlockyTalky code and evaluate the musical correctness of the sentences.

Participants showed negative self-efficacy, specifically in accomplishment or performance outcomes, when expecting the results of instructionalist feedback (right or wrong) while presenting understanding to graduate and professor participants. The negative outcomes of this self-efficacy included disengaging during group share outs and overfitting to the perceived instructions of given examples.

The takeaway for instructors is to be mindful of the knowledge that students build collectively and by what acceptance criteria they consider progress and success. How instructors both query understanding of the computational model as well as provide illustrative examples might overwrite the opportunity for student agency causing students to erroneously interpret feedback as corrective and overfit their designs. As a suggested generalization, when providing student agency like this with a broad application domain, imposing more restrictions on the application domain (like artificial technical limits) may help. These limits can be lifted as students encounter effects of their limitation during iteration as computational goals are met. This project ended in a performance which instead used time for rehearsal that would usually be spent on iteration in a typical CS project classroom. In these rehearsals, participants practiced the song with the running program several times to perform it correctly, but no changes to the code were made between practices. Rehearsal solidified what the runtime behavior was which is valuable for self-efficacy since participants associate the resulting project with success and usefulness. In viewing just the final product, this insight would be missed and future work should study the relationship between iteration and rehearsal (or equivalent) of student designed projects where preparing a presentation of the runtime is a component.

4.3 Warble

We designed and implemented Warble: a textual, Haskell embedded DSL for implementing




```
[warble|
  component DigitalSensor port
  (- output :: Int -)
  [- driver :: Int -]
  {
    setup output <| 0 |>
    setup driver <| initDSensor |>
    merge output <| mergeDSensor |>
    tick driver <| tickDSensor port|>
    [- driver ]- id >( output -)
  }

  component BlinkingActuator port
  (- input :: Int -)
  [- driver :: Maybe Int -]
  {
    setup input <| 0 |>
    setup driver <| initDActuator |>
    merge input <| mergeDActuator |>
    tick driver <| tickDActuator port|>
    (- input )- Just >[ driver -]
  }

  universe BlinkingLED
  {
    b :: DigitalSensor <| GP.D4 |>
    led :: BlinkingActuator <| GP.D3 |>
    (- b output )- id >(led input -)
  }
|]
```

Figure 4.9: An example, simplified Warble diagram showing dataflow between two hardware components parameterized by which hardware port they connect to. In text, the equivalent Warble DSL provides an implementation with added detail about the internal state of each component.

dataflow programs inspired by PureData [Puc+96] between stateful hardware components based on propagator logic described in Alexey Radul's thesis work [Rad09]. The concrete syntax for quotation and antiquotation followed conventions from other Haskell embedded DSLs, specifically the PADS project [FW11]. While not intentional, the component-and-arrow syntax resembles Haskell syntax for a computational model called Arrows [Pat01]. In comparison to BlockyTalky, Warble has a small application domain (wearable computing) that focuses on sensors and actuators, but does not include the explicit creation of messaging protocols and events are abstracted as either the values polling hardware produces or transformations on those values.

From these restrictions, Warble implementations match a visual metaphor of describing dataflow programs with bubbles and arrows. We took as our hypothesis that the visual metaphor would facilitate teaching the computational model and let students make design choices to exercise their agency, and avoid overfitting particular examples instructors provide. We intended for visual correctness (a bubble for every hardware component and an arrow for every data dependency) to be the acceptance criteria shared with and between participants. Then, once participants were happy with the diagrams, each piece of the diagram could be represented with DSL code.

Related work on function design with first-time programming students at the university level splits TA and student interactions into five tasks, and TAs spend significantly more time in two of them: (1) understanding the data parameters and return values of function as well as (2) the function's implementation details [RKF19]. By designing Warble programs with the visual metaphor, the functions students design are small computational units connected in a larger context the students create which should be easier to implement as well as give purpose to determining the forms of data for the parameters and return values.

4.3.1 Warble: a technical introduction

We chose Haskell as Warble's host language to use the library Template Haskell [SJ02] as a metaprogramming environment and the Parsec library as a parser. Haskell's type system and its syntax were borrowed to describe and validate hardware components in Warble.

To highlight this visual metaphor and borrowing of syntax, figure 4.9 contains an example diagram that illustrates a wearable where a digital sensor, like a button, controls a blinking actuator, like an LED and the corresponding Warble DSL code.

In the diagram, the digital sensor component is represented with a circle and can produce data from a connector labeled output with type Int. The sensor component takes a parameter, port, as metadata of what hardware port the component connects to. The square box represents a driver representing the hardware that performs IO operations. A component that can produce or consume values purely logically does not need a driver, but we use a square to represent that the driver can produce or consume data from outside the diagrammed system. The blinking actuator component is defined very similarly except its driver consumes `Maybe Int` values instead of `Int` values.

While not important for understanding the computational model, this program will use `Maybe` data constructors in its driver implementation to decide when to not call IO unnecessarily. Specifically, the hardware used to test Warble treats sensors and actuators as file descriptors where

the program can save system calls by not writing unchanged data.

An arrow shows the dataflow of the system from one component's connector to another and represents the host language transformation function to convert values. In this case, the output values can be equal to the input values so that when the button is pressed the LED starts blinking and when the button is released the LED stops blinking. For this logic, we use the host language identity function from Haskell's standard library `id :: a -> a`

The code implementation in the Warble quasiquotes (with syntax `[warble| ... |]`) shows reusable declarations for these components using the keyword `component`. After the keyword, the component's name and parameters are declared with Haskell's type identifier syntax chosen for names such as `DigitalSensor` and variable identifier syntax chosen for host language variables like `port`. As part of the signature, we declare pure logical connectors using `(- x :: Type -)` syntax and side effecting IO connectors with `[- x :: Type -]` which match the overlayed round and square shapes representing connectors and drivers on the periphery of the component circle in the diagram.

Scoped by curly braces, a component defines its implementation details using the keywords `setup` to initialize the component connectors, `merge` to update the state of the component connectors with respect to new values, and `tick` to specify behavior as the runtime system advances. Host language expressions are used to implement these details and are spliced using antiquotes `<| e |>` where `e` is a Haskell host language expression. In the last line of the component implementations, the logical connector and the hardware connector are attached to show how data flows in and out of the system using IO.

Informally, we use the term system to describe parts of a dataflow program including components and their connectors. Formally in the DSL, there is a keyword `system` to describe a higher-order component that can be parameterized by connectors and contain subcomponents (not shown in figure 4.9).

A universe is a specialized system that has no component parameters or declared connectors in order to be self contained and can be declared with the keyword `universe`. Since a universe is self-contained, a host language IO program is generated from a universe declaration that can be run using the Haskell runtime system and the Warble standard library. For example, from the universe declaration `BlinkingLED`, a host language value `mkBlinkingLED` is defined in scope. This value can be invoked, like in the expression `System.Warble.run mkBlinkingLED`, and has type `IO ()` typically to be used from the outer program's main function. In the implementation for this universe, an instance of the sensor and actuator components are declared using type notation and the particular ports for these components are passed as host language values from a library `GP` using antiquotes. The identity arrow connecting the components in the system diagram is then implemented using the notation `(- b output )- id >(led input -)` which can be read as transform values from the connector `output` on the declared sensor component `b` using the identity function and give them to the `input` connector on the actuator component `led`.

Both side-effecting drivers and logical connectors generate code for propagator abstractions, variable-like state machines, to hold values. The transformation arrows between connectors and drivers form propagator dependencies. In the generated code, when the `tick` function

is called by the runtime system, values are stored in the drivers' propagator abstraction and the dependencies handle transforming the values and populating the listening propagators if the value has changed as determined by the connectors' and drivers' `merge` functions. These propagator abstractions are provided by Ed Kmett's Haskell library implementation of propagator logic [Kme15].

4.3.2 Computational Model

The computational model for Warble focuses on the PL concept of abstraction through the use of defining and naming components, systems, and universes. Teaching this model uses an impovrished view of functions as simply transformers of values from one type to another, but relies on function currying to be able to customize implementation details. For example, the `tick` definition partially applies the curried functions `tickDSensor` and `tickDSensor` to the parameter `port`.

As an implementation detail of the runtime behavior in the computational model, connectors are viewed as stateful. Transformations between connectors are effectively listeners on changes to the source connectors' state. When the source connectors' state changes, the transformation applies the function to the changed value and sends the resulting value to the recipient connector to merge with its current state. These changes potentially propagate through the dataflow system and can be thought of as a restricted message passing model where the protocol is derived from connections. Two systems are concurrent if they are unreachable from each other.

4.4 Case Study: Poster Design Activity

4.4.1 Context

Seven 11-14 aged first-time programmer participants were randomly assigned from a population of summer camp attendees to a workshop for making wearable devices with Warble. The workshop was held over eleven 90-minute sessions. Each session would begin with a small demonstration and an unplugged-style activity with the intention of soliciting participation and constructing shared understanding with the group before validating that knowledge using the computer. In these demonstrations, instructors would *live code* for the participants to illustrate a new computational concept as it would appear in the Warble programming environment. In the unplugged style activities, instructors would have the participants act out or demonstrate how the computation from the demonstration would behave without relying on the computer to check. By making participants act as computational processes, they would build an intuition and form questions early in plain language because they could not put-off demonstrating their understanding until implementation details, like compile-time errors, were satisfied.

As an example, the computational idea of a function transforming values of one type to another was illustrated with an unplugged activity where participants acted as the functions, sticky notes acted as incoming and outgoing values, and the participants used pens to convert each value to another for given expressions. Say participant Adder had the function `plusOne x = x + 1` and participant Duffy had the function `minusTwo x = x - 2`. If the instructor showed the expression `(plusOne (minusTwo (plusOne 5)))`, the instructor would write five on a sticky note, hand it to

Adder who would cross five out and write six, Adder would hand the note to Duffy who would cross six out, write four, and hand the note back to Adder who would cross four out, write five, and finally show the class the note with five as the final value.

4.4.2 Data set

In this case study, we look at one session's unplugged activity where participants showed their understanding of the computational model by drawing a poster of the device they designed, placing hardware sensors and actuators physically on the poster, and drawing the computation using the visual metaphor, like in figure 4.9, that instructors modeled during the demonstration. This data set provides data from five of the seven participants who worked in two groups: a pair and a triple. One of the other two participants was absent and the remaining participant collaborated with an instructor, so their data and results are omitted from analysis. Participants were asked to indicate areas on the illustration where they could put the hardware they would be using and to attach the hardware to the design poster if that would help them plan and discuss. Instructors modeled the activity for the running example of a mood ring where integer readings from a heart rate sensor were transformed into enumerated values representing abstract states of emotion like happy and sad. To ask students to think computationally at the type level, instructors asked the participants to annotate the arrows on the dataflow diagram with Haskell types if they could. Instructors expected function types of the form `:: a -> b`, to be annotated on the arrows that connect one component to another.

We provide survey data originally intended to qualitatively assess Warble as an accessible DSL for the application domain of wearable computing. But participants notes on the workshop expectations and their successes provide information of self-efficacy.

4.4.3 Poster artifact

In Figure 4.10, the participants illustrate a design for a smart umbrella that detects inclement weather using the ambient humidity, wetness, wind, and temperature. The device notifies the user of the weather and would automatically open and close under certain conditions. The primary feature of their design is the placement of the humidity sensor hardware on top of the umbrella, noted with a drawn box for where they put the sensor during prototyping. On their computation diagram, they use circles to represent these sensors and actuators with data flowing from sensors to actuators. The humidity sensor sends data to the phone (display) actuator with a transformer from the integer humidity value to a displayable string representation. The wind (sound) sensor and moisture sensor are both connected to a logical device labeled plastic which refers to the body of the umbrella. They decided the *plastic* component to be the logical controller deciding when to open and close the umbrella. This logical component connects to a motor to perform the opening and closing actuation. The group ran out of time before they finished the computation diagram; the output of the wind/sound sensor, the input types of the logical plastic sensor, and the output type of the logical device were all left unspecified. The humidity display system and the opening and closing system are assumed to be concurrent as they are not connected by transforming

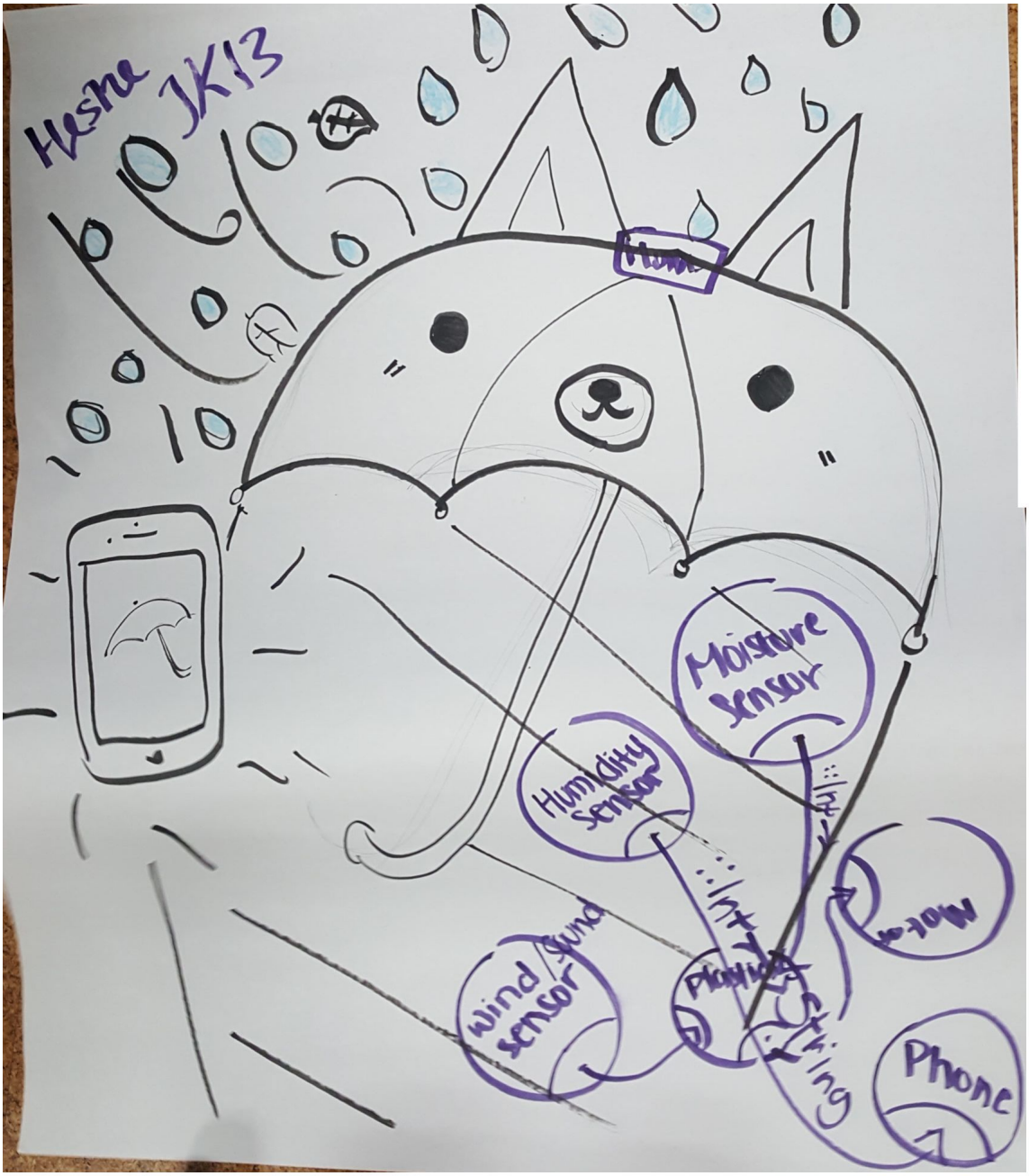


Figure 4.10: The design poster for a smart umbrella as a wearable program to be implemented with Warble. The participants made an illustration to help solidify the specifics of the project. They highlighted hardware components with a square box on top of the umbrella and a phone abstraction. A bubble and arrow diagram shows an intended universe for their program with the components labeled and connectors connected with types.

arrows.

In Figure 4.11, the participants illustrate a design of a wearable watch that tells time and shoots lasers. To help understand how they would put it together, the participants taped the button sensor, display actuator, and a crayon representing the laser actuator to the poster. As a result of this illustration, instructors learned that a physical laser actuator and a logical time sensor needed to be built for the participants to be able to implement their project easily. In their final project, a commodity laser pointer was attached to a generic digital actuator to form the laser actuator and a time sensor was simulated logically as an IO call to the wearable device's OS API.

In their computation diagram, the participants specified that the time sensor would produce integers (unix timestamps were verbally discussed) and must be converted to strings for displaying on the LCD display. The button would produce integers (0 or 1) and the laser actuator would consume integers (0 or 1) for corresponding on and off states. The time and laser systems are concurrent since they did not share connections.

4.4.4 Resulting code

Student's wrote code in a graphical terminal text editor using keyboards and monitors connected directly to Raspberry Pi boards. These boards ran the Haskell and Warble compiler and runtime system locally and did not rely on internet connectivity (including local internet). GrovePi brand i2c capes were attached to each Pi to connect to hardware sensor and actuator components. If students saved their programs in a particular directory, the programs would run on startup. By running on startup, the wearable devices could be disconnected from the keyboards and monitors and tested using battery power packs. Students demonstrated their final projects at the end of the camp by presenting the wearables in front of camp attendees from other workshops and classes.

```
[warble|
  universe Umbrella
    {
    hs :: TempHumiditySensor <| D2 |>
    ld :: LCDDisplay
    ds :: DigitalSensor <| D3 |>
    as :: AnalogSensor <| A0 |>
   -- :: Plastic
    mo :: Motor
    (- hs humidity)- show >(ld text -)
    (- ds output )- control >(mo input -)
    }
|]
control :: Int -> Int
control 0 = 0
control 1 = 100
```

Figure 4.12: The finished code for the smart umbrella wearable project with a DSL quasiquoter containing the code corresponding to the participant's design diagram and the host language Haskell function `control` the students had to write to implement the behavior.

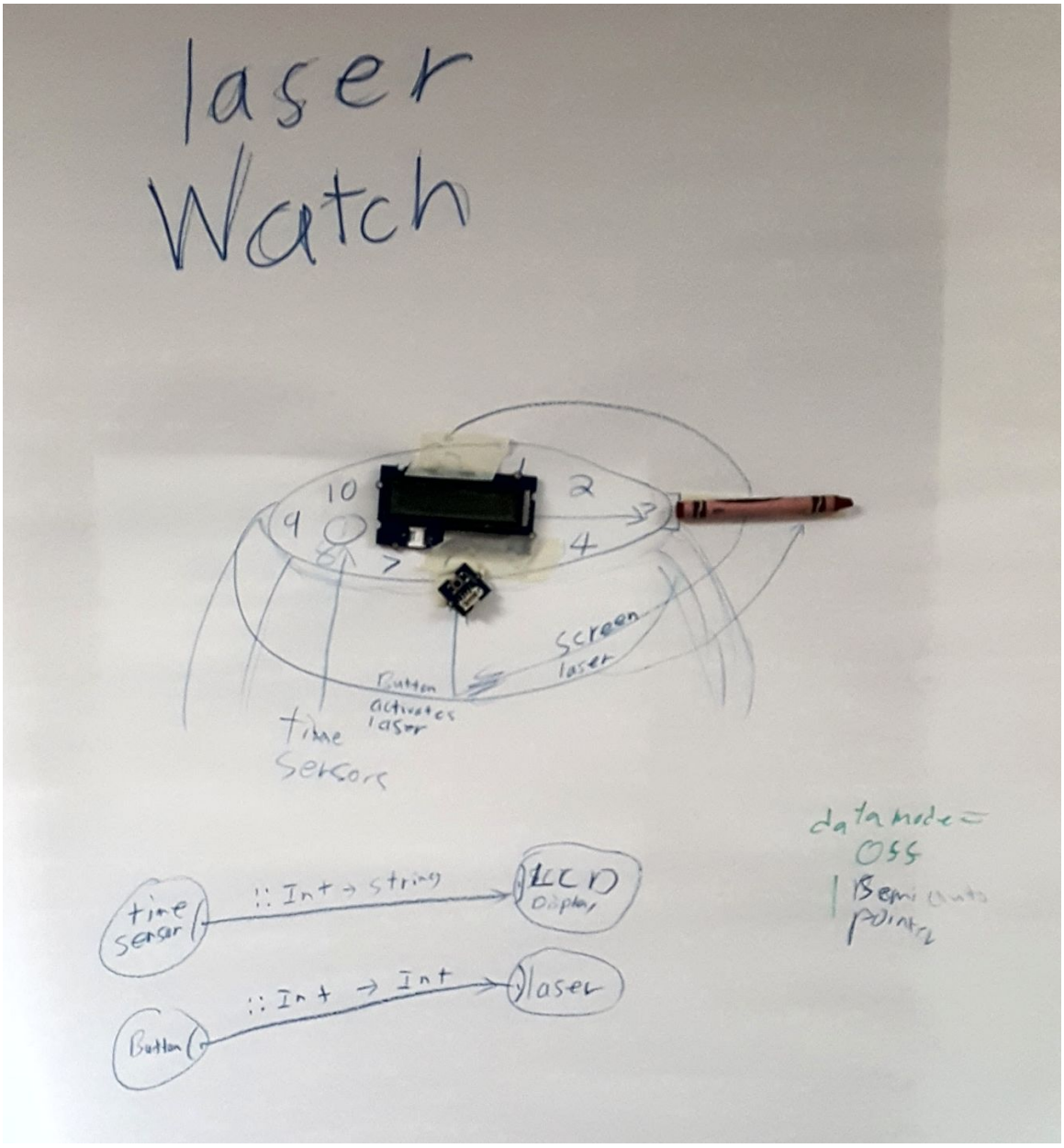


Figure 4.11: The design poster for a wearable that lights up when a button is pressed and displays the time. The participants made an illustration and attached the hardware of three components to it: a button sensor, an LED display actuator, and a crayon representing a laser pointer. The bubble and arrow diagram shows an intended universe for their program where two sensor and actuator systems are connected in parallel.

Figure 4.12 contains the code for the participant's smart umbrella project inside of the quasiquoter syntax declaring the DSL scope: `[warble| ... |]`. Their project used prebuilt sensor and actuator components from the Warble standard library and consists of a closed system declared using the `universe` keyword and a host language Haskell function they authored, named control. From the Warble standard library, the **`TempHumiditySensor`**, **`DigitalSensor`**, and **`AnalogSensor`** abstracted over hardware components plugged into physical ports. To specify the physical port, these library components use antiquotes `<| ... |>` which take a host language Haskell expression to pass parameters to the generated Haskell code for the Warble library component. For these particular components, physical ports were represented with Haskell data constructors used in the library wrapper for the GrovePi hardware. For the components **`LCDDisplay`** and **`Motor`**, their implementation in the Warble standard library were mapped to statically known file descriptors so they did not need parameters to initialize or run.

The participants simplified their project by not using a moisture sensor to determine inclement weather and instead used a generic digital sensor corresponding to a button to decide when to open the umbrella. From this decision, we realize that the design poster did not capture all of the implementation details needed for them to simply translate their designs into host language code with respect to the wearables application domain, but they were able to easily swap out another implementation still using the dataflow computational model. Their program pipes the humidity output of the temperature and humidity sensor plugged into port **`D2`** on the hardware device to an LCD display to simulate the phone actuator their originally designed. The declarations of transformations between connectors use the syntax `(- e1 )- f >( e2 -)`. `e1` and `e2` are host language Haskell expressions that must evaluate to connectors of warble components. `f` is a host language Haskell expression that must evaluate to a function that converts a value from the type produced by the connector `e1` to values of the type consumed by the connector `e2`.

In this program, `(- hs humidity)- show >(ld text -)` takes integer values from the humidity connector of the **`TempHumiditySensor`** component initialized on port **`D2`** with local identifier `hs`. It uses the host language polymorphic function `show`, part of the **`Show`** typeclass, to convert the float value to a string. Then it sends the string to the **`LCDDisplay`** on its text input connector.

Similarly, The button input of the **`DigitalSensor`** component is converted to a step-motor value `100` which operates the motor far enough to open the umbrella as implemented in the host language function `control` that the participants' authored.

```
[warble|
  universe LaserWatch
    {
    led :: DigitalActuator <| GP.D6 |>
    lcddisplay :: LCDDisplay
    b   :: DigitalSensor <| GP.D2|>
    ts  :: WallClock
    (- b output )- id >( led input -)
    (- ts output )- id >( lcddisplay text -)
    }
|]

[warble|
  component WallClock
  (- output :: String -)
  [- driver :: String -]
  {
  setup output <| "" |>
  setup driver <| return "" |>
  merge output <| mergeIfChanged |>
  tick driver <| tickClock |>

  [- driver ]- id >( output -)
  }
|]
```

Figure 4.13: The finished code for the laser watch wearable project with DSL quasiquoters containing the code corresponding to the participant's design diagram and a custom component.

Figure 4.13 contains the code for the participant's LaserWatch project inside of the quasiquoter notation `[warble| ... |]`. They use the `universe` keyword to declare their system and define a custom component using the `component` keyword. Similar to Figure 4.12 containing the smart umbrella, this code for the LaserWatch declares components for sensors and actuators. In their project, they declare four components that match the four on their design poster. In their dataflow connection declarations, their implementation simplifies the logic more than the poster. In figure 4.11, the connection between the time sensor and the display actuator was expected to convert unix timestamps represented by integers into strings. Instead, the participants authored the time sensor as a component called **`WallClock`** which produced time output as string values in human-readable form. By doing so, they were able to simply use the host language Haskell identity function `id`.

To design the **`WallClock`** component, the warble DSL program must declare the connectors of the component using the notation `(- id :: Type -)` which take a host language Haskell identifier and type. Also, the sensor component must declare the logical driver to specify how values are produced from outside the program such as with IO and hardware (or consumed in the case of actuators). To initialize the connectors and drivers, the Warble's DSL keyword `setup` is used, given the identifier of the connector or driver, and then given a Haskell host language expression in antiquotes `<| ... |>` to store the initial value. Connector values must be pure and can be pop-

ulated with pure host language expressions. Driver values may be impure so they are populated with monadic host language expressions.

For example, in the `WallClock` component, the driver is initialized with an empty string wrapped in a monadic context determined by the Haskell type system (such as Identity,IO, etc) from the use of the function `return`. Then, the `merge` keyword specifies a logical update on the output connector. Specifically, the host language function parameter to `merge` takes the old value stored on the connector and a new value sent to it by a connection to decided what value, if any, should be saved and sent to downstream connectors. In the `WallClock` component, the output connector only sends new values if they are semantically unequal to the current value using the Warble standard library host language function `mergeIfChanged :: (Eq a) => a -> a -> Maybe a`. For example, if the port was given a string timestamp that was equal to its current value, it would evaluate to `Nothing` and prevent downstream dataflow computation of redundant values. Otherwise, it would pass along the newly given value and overwrite the previous one. The `tick` keyword is parameterized by how it will produce a value for every pass through Warble's runtime system's main loop. In this case, it calls a host language Haskell IO function written by instructors at the request of the participants to ask the operating system for the current time as a string.

4.4.5 Survey Data

Did Warble enable you to express the big ideas from your project?	If you were a youth leader next year, what would you keep?	What would you change?	How do you now feel about Wearable computing? (Is Warble a good choice?)	Any other suggestions?
Yes, My ideas	I would keep friends	change bad things	Good, because it is fun	No
Yes	I would keep the part where we explored and used different sensors / actuators.	I would like more time to do the project	It helped during the building/coding process	
Yes, my ideas	the project ideas		Yes, because it helped me find new skills	
Yes, maybe not actually finish the project, but almost done	the friendly environment		Yes, because it wasn't over complicated	Always make sure that everyone is on the same page or no one is behind
Yes	Same project ideas	more take home materials		Thank you!
Yes, how sensors/actuators work	project and demonstrations we did		It was challenging and fun at the same time	

Table 4.1: The post-camp warble survey results

Table 4.1 shows survey results from six out of seven of the participants (one had left the camp before the last day). The first row shows the qualitative questions asked to get feedback on if Warble, as a DSL, was an appropriate tool for them to use.

For understanding their self-efficacy, this survey is congruent with instructor opinion that the project and methodology produced a successful social learning environment. Of particular interest, several participants noted the friendly environment as something they would keep if they were to teach Warble to others in a similar camp environment. Laments of the workshop included the expected constraints of time management and the inability to allow participants to take home hardware materials. The simplicity of the workshop content was acknowledged, and difficulty was framed as "challenging" with repeated emphasis on being able to represent project ideas and understand particular concepts about hardware.

4.4.6 Warble Results

By adapting the goals of the BlockyTalky study's design activity to a textual DSL with a less expressive computational model, the participants in the Warble study were able to exercise their agency as project designs using the visual metaphor. Specifically, participants were given framing agency towards the end goal of the project and the content and degree of the logical operations needed to implement the project.

Unlike the BlockyTalky study, we fixed the activity instructions so that participants had to encode and discuss their ideas using a diagramming structure that had a particular syntax. Code that corresponded to the metaphor enabled participants to represent their ideas in terms of the computational model with the intent of implementation being mostly a translation from the diagrams to text. The participants had agency over the end goal in terms of satisfying criteria for their projects, i.e. what the final project should be able to do. The participants had agency over the logical operations in that they could choose how many components (bubbles), how many transformation functions (arrows), and what overall algorithm (connectedness) they needed to achieve those satisfying requirements. From this agency, we intended for participants to determine what would be good for the project. We found that, like the BlockyTalky study, students exhibited epistemic agency for intrinsic value when translating their diagrams into code that simplified the project, but met the criteria they decided, i.e. their code differed from the visual metaphor while still performing behavior equivalent to what they illustrated. The designs were simple, making them easy to evaluate, implement, and test, but without observations of participant discussions, it is unclear whether the simplicity of the designs means no agency for learning was had, i.e. were the participants' partial understanding presented and worked out using the diagramms. From student feedback, this lack of agency for learning did not hinder self-efficacy of accomplishment and proficiency outcomes, but may be apparent in the participants' requests for more time to iterate on their projects. It would otherwise be unclear why participants' would want more time when their designs met the shared acceptance criteria and their projects met their satisfying criteria.

From the perspective of epistemological messaging, the visual metaphor and less expressive computational model helped us separate the content and form of the computational thinking from the production of it. Concretely, instructors were able to focus on helping participants un-

derstand if a design was possible in the computational model before helping participants with the technical knowledge needed to write code implementing that design. However, it is unclear if the visual metaphor caused participants to avoid designs requiring complex implementations since the designs were visually simple.

The student-agency-focused teaching interventions promoted engagement and collaboration by instilling intrinsic value of the project in participants but exhibited the recurring challenge of avoiding the breadth and depth of computational thinking using the computational model. Conversation video data is unavailable for this study to show the ownership of knowledge amongst groups and how these design decisions were formed; instead, we use the survey results to approximate student impressions and their self-belief. The survey results support the idea that students believed they were given agency over their design "ideas" despite the imposed form and acceptance criteria of the visual metaphor. We would like to discover if either the DSL or time limitation was the primary source of design simplicity since participants noted time limitations in their surveys, but reported success with the computational model and expressiveness of the DSL. For the smart umbrella, the switch from sensing inclement weather with a relevant sensor to a button fulfilled demonstration constraints; the group did not want to utilize water or wind inside the building for fear of making a mess. It is unclear how compromising their decision this way affected their self-efficacy through proficiency. I.e. did that compromise make this not a successful experience? However, we believe that the vicarious experience self-efficacy outcomes for the participants were positive from the mentions of the friendly environment, general positive feedback, and attentive group participation despite no extrinsic rewards (there were no grades and students were not able to keep their projects).

4.5 Discussion

The BlockyTalky study shows how first-time programmers reason about PL concepts in the application domain of musical performances, how the design metaphor can solicit corrective feedback, and how the transition from knowing what the project is to how the project will work impacts self-efficacy. Specifically, students spoke readily in the application domain of musical performance, but abstracted away parts of the computational model. In terms of self-efficacy, participants created an unspoken, shared acceptance criteria of syntactic and semantic correctness in the application domain mapping to some semantic correctness in the computational model. For example, if a user story is grammatically correct and the musical performance vocabulary used makes sense, then the code to implement it should work in some way. High self-efficacy determined who felt confident to help each other and accept roles in their groups. However, when asked by instructors rather than peers to explain the semantic meanings mapped to the design results, almost all students interpreted the request as corrective and became unable to provide a corresponding example in the computational model. Students anticipated instructors would provide the correct answer when multiple viable solutions could exist for their user stories.

The Warble study shows a potential refinement to the unplugged activity. A smaller application domain using a less expressive computational model with visual metaphor may improve

applying student designs to the computational model while maintaining student agency and positive vicarious experiences. The primary challenge remains: how do we encourage exploration and depth of learning when project *doneness* is determined by student chosen project designs? Specifically, teaching interventions should provide acceptance criteria for complexity, but without cognitively overloading students. In considering epistemological messaging and participant feedback, we should explore how iteration and time management techniques in these projects could help [Rus18]. That is, allow initial designs to be simple and foster intrinsic value in the project, but add external constraints later on to focus on more ideas and the relationships between them.

In both cases, the use of a DSL successfully enabled participants to make projects of their own design and abstracted away many representational issues first-time programmers face that get in the way of computational thinking. Positive proficiency self-belief and vicarious experiences were enabled by the application domains since they allowed participants to approach programming concepts from an intuitive context they already had like musical performance or wearable product drawings. Unplugged design activities enabled student agency and provided insight on where instructor intervention and peer discussion can improve or damage that agency.

Non-code artifacts provided insight into self-efficacy with notable examples. There exists a perceived change in difficulty and ability when projects become things that need to be delivered (from a non-realized to a realized project). Presentations and perceived difficulty can be seen as threats to proficiency self-belief in preparation but also as support after successes. In the absence of acceptance criteria, learners will invent ones and rely on the affordances of the activity to correspond to correctness in the computational model. The omission or abstraction of ideas in non-code artifacts, like missing diagram annotations or explanations that remove computational elements, provides insight into the thinking students have of the computational model, which may then manifest in the code implementations as less-than-productive usage (such as the long motifs with BlockyTalky or the simplification of projects with Warble).

If education shifts towards introducing bespoke programming languages into non-programming education domains, such as with teaspoon languages [Guz21], then design activities and tools must consider the factors of student agency as input, student attention during facilitation, and student self-efficacy as output to be successful. These factors will impact non-programmer educators' success using such tools and the far transfer of the knowledge non-programmer or first-time programming students gain by participating in those intervention.

5

Self efficacy and participation in introductory PL recitations

5.1 The course

We taught a required Programming Language (PL) course for undergraduate sophomores to early graduate students. In the homework for this course, students were asked to demonstrate proficiency in new PL topics each week. Each topic could introduce new syntax, semantics, styles of programming (e.g. continuation-passing, object-oriented), or methods for solving problems such as data-driven function design using proof systems, operational semantics understanding, or proof writing. When encountering difficulty on the homework, students exhibit confusion in one or more of these topic areas. Because students don't know what they don't know, teaching assistants (TAs) spend office hours helping students determine if they are misunderstanding syntax, semantics, style, methods, or some combination which is a challenge in epistemological messaging.

5.2 The issue

Students typically start asking for help once they have concrete details of work they have produced so far: partial answers to homework problems such as code. Following their training, course TAs make an effort to guide conversations towards understanding the underlying concepts that will help the student on the entire homework assignment and beyond. TAs experience more difficulty guiding these conversations as the homework deadline approaches. The system in which students queue for help from a TA reveals that students usually wait until near deadlines before asking for help in office hours.

When necessity forced TAs and students to switch to online-only teaching modalities, TAs encountered extra difficulty because they lost the collaborative communication that came from holding office hours in a shared, physical space. Students waiting for their turn did not get the benefit of hearing other students' questions and the TAs' responses. Students fell into a negative feedback loop where ineffective questions or detailed requests made help take longer, increased wait times for future help, and incentivized students to queue themselves for help even before they

had concrete questions. These issues only got worse as the semester progressed into more difficult materials where students who did not need help earlier in the semester started joined the office hour queues.

To alleviate this pressure, we wanted to change our teaching methods to help students discuss PL concepts more effectively earlier in the course.

5.3 Goals of the intervention

We designed our teaching intervention around the first problem of each recitation, which are TA-led, group labs of up to 15 students. In the intervention, we asked students to critique partial answers, from imaginary students and TAs, that modeled syntax and illustrated what the recitation topic was asking. By focusing student attention to comparing and contrasting the partial answers, students would be better prepared to talk at the conceptual level. We hypothesized that removing the right/wrong aspect of the initial interaction could positively affect student participation by lowering the social cost to collaborating. With more participation at the beginning of each recitation, TAs would discover potential misunderstandings more readily and fix confusion before it manifested in office hours.

By being better prepared to discuss concepts, students could talk with their peers and TAs earlier, ideally *before* encountering issues on the homework.

5.4 Contributions

In both an online and an in-person offering of the Programming Language course, we modeled syntax and problem decomposition in the first question of each week's recitation handout. We investigated whether this intervention positively impacts participation during the recitation and improve how prepared students are to discuss the material. Our desire to investigate student participation and self-efficacy took inspiration from similar goals and positive outcomes in an Algorithms course study by Enström and Kann [EK17]. Support for applying our intervention where TAs help students build initial concepts, like the beginning of recitation, comes from Ren, et. al. on how TAs spend their time helping students [RKF19].

Our intervention models syntax and problem goals to help the TA direct student thinking to concepts in an effort to reduce the time TAs spend in problem formation as an application of epistemological messages. To maximize the benefit of this modeling, TAs were instructed to facilitate explanatory feedback following Moreno's work showing low prior knowledge students benefit more with explanatory feedback than with corrective feedback.

We randomly assigned each recitation to use either control (A) or experiment (B) handouts. In Handout A, students solved problems from a blank slate. In Handout B, students instead critiqued partial solutions for the first problem, and answered blank-slate questions just as in handout A thereafter.

After each week's recitation, both students and TAs reported whether they perceived participation was less than, more than, or as expected in six categories. Students also ranked six tasks

from most to least prepared on that week's topic, three discussing it (1) with a peer, (2) with a TA, or (3) on the class forum, and three applying it by (4) studying it with the book and handouts, (5) writing code, or (6) debugging programs. We discuss the design and results of this survey and the selection of the statistical tool, the Mann-Whitney U test, in section 6.6.

From the results, we conclude the effectiveness of our intervention, consider TA impressions from post-experiment interviews with TAs, discuss what confounding effects might impact our conclusions, and plan for future study. We intend for this research to contribute to studying mid-level courses that integrate design and theory components with coding like PL and Algorithms. We benefited from the growing research into how social effects and self-efficacy make an impact on student learning in these areas and want to contribute to helping educators work smarter and at scale in content heavy courses resulting in little opportunities for student agency.

5.5 Theory basis

In considering the problem TAs encounter with respect to epistemological messaging, TAs would like to approach interactions with students by communicating the production of the ideas. Specifically, TAs would like to direct their attention to help students understand how to apply the concepts from the lecture materials with a shared understanding of the form of the ideas the homework is asking. We notice students often do not have the content of the lectures readily memorized causing TAs to focus on the content of ideas. Students also need an example showing the form of the solution they are being asked to produce since several forms are used throughout the course. When students divide their thinking among these categories, they experience an increase in cognitive load and TAs experience difficulty applying their help to the entire group because of the divided attention.

In an effort to design a question to reduce cognitive load by exemplifying the content and the form of the topic, we turned to the cognitive science literature. Moreno conducted a study testing whether explanatory feedback or corrective feedback was better in retention and far transfer (application to new problems) [Mor04]. While there is no significant difference on immediate retention of concepts, explanatory feedback was better at far trasnfer where students would apply the concepts to new problems. They partially confirmed their hypothesis that "Low prior-knowledge students learn best from discovery-based multimedia environments when pedagogical agents guide them with explanatory feedback during the process of meaning making. Explanatory feedback reduces the extraneous cognitive load that results from having novice students search for a plausible explanation to the correctness or incorrectness of their problem solutions in discovery environments." While our PL course was not exactly the same discovery environment, we decided to design our problem such that our pedagogical agent (in this case TA-led instruction and the handout) would be more inclined to explanatory feedback rather than finding the first correct answer. We hoped that this would alleviate TA frustration with students not being prepared to begin recitation (not watching the lecture), and that the far transfer effects would be present in how prepared students are to discuss topics in office hours.

5.6 Methodology

In our experiment and survey design, we prioritized minimal intervention to improve participation in recitation and illuminate how prepared students believed they were to discuss and do work on each topic. When we made design choices, student and TA stress avoidance was always most important given the intensity and time commitment of the course as well as the semester-long nature of our intervention and measurements. We kept the intervention small by only changing the first recitation question, but the coverage of the study broad by doing so for each recitation. Unlike Enström and Kann's study in algorithms, we did not have a particular topic that stood out above the rest in difficulty, and so we hoped this small, but broad approach would inform future areas of study, like which topics to focus on. While the downsides of this approach include limiting the types of data we collected, limiting interaction with student participants, and asking more general questions, the unobtrusive nature of the study was easily fit into the expected service of the course for improving teaching practice and TA training, allowing us to be reviewed and category-one exempt under our university's internal review board.

5.6.1 Participants and course context

All students who elected to take the intro to programming languages course were required to participate in recitation activities. As part of typical course work, where we pursue internal improvement to the course, the handout designs and weekly surveys were given to all students and TAs.

From this data, we can abstract a view of our participants, however TAs and students freely changed between recitation sessions week-to-week and were absent as a normal part of the educational environment. So, the participants of our study included a maximum of 104 participants of mid-level university students during the online semester and 84 students during the in-person semester. We had 22 TAs during the online semester and 18 TAs during the in-person semester. Full participation numbers are available in table 5.2, and the drop off in participation can be attributed to the length of the study and the acceptance criteria of participation. Specifically, students did not need to attend every recitation nor submit every survey to receive a good or passing participation grade. TAs were simply encouraged to participate for the amelioration of the course's structure and perspective of their teaching practices without any detriment or reward that would affect their employment or professional evaluation.

These students were required to have taken the introductory computer science courses, a systems course equivalent to university level C-language and computer architecture courses, and a discrete mathematics course as prerequisites. TAs were chosen from students who volunteered to be TAs, had taken the course before, and had done moderately well or better in the course at the professor's discretion. These students were mostly second or third year computer science majors or first year graduate students, but some students could have been further along in their program. It is highly unlikely any students would have been earlier in their university careers.

The online course and in-person courses were both conducted during the COVID-19 pandemic which placed external pressures on TAs, students, and professors. However, the online

lecture materials for the courses were recorded prior to the necessities of the pandemic and the online classroom environment had been run on an online class during the pandemic before the semester of this study. The experimental recitation learning materials were crafted and refined in the semester prior to this study. The online semester prior to this study similarly held the recitation over video conferencing software where students get put in breakout rooms to simulate pair discussion.

The in-person semester in this study was performed in the semester after the online semester, where the pre-recorded lecture videos were still used, but recitation labs were held in physical classrooms and students were encouraged to talk to their neighbor during pair discussion instead of methods similar to isolated breakout-style rooms.

5.6.2 Handout and Experimental Design

We randomly assigned each recitation to use either control (A) or experiment (B) handouts. In Handout A for a given week, students solved problems from a blank slate. In Handout B, students instead critiqued partial solutions for the first problem and then answered blank-slate questions, just as in Handout A, thereafter.

When enrolling for the course, students signed up for recitation sections based on scheduling constraints. Subsequently, we randomly assigned handouts to sections based on enrollment numbers to ensure roughly half the students worked with each handout type. Students could choose to attend a different recitation section in a given week to make up attendance and so could experience both A and B style handouts in the same semester, but not for the same recitation topic. Students knew that the handouts were labeled A and B and that we were conducting experimental research on teaching methods. They further knew that the two types of handouts were of equally high quality, and they could revoke their consent to participate in the analysis at any time, although they still had to complete the surveys as part of their feedback for participation in the course.

Table 5.1 outlines the high-level transformation of the first question of handout A into the first question of handout B. The first column is the week's topic organized chronologically. The second column describes what handout A's first question asks students to do. The third column summarizes the expected solution to that question, which is provided to students in a solutions handout at the end of the recitation. The fourth column sketches what handout B asks students to do instead. Specifically, it presents the same question from handout A, introduces imaginary actors like students and TAs, and then asks recitation students to discuss what they like, did not like, agree with, or do not agree with in the imaginary actors' answers, comments, explanations, and questions. The imaginary actors always model the new vocabulary for the topic, always write syntactically correct answers, and always address the form of the question. The imaginary actors may make conceptually incorrect comments, vague comments where it is unclear if they have the right answer for the right reason, or simply present a question that recitation students can answer or echo. In the course, we use a *Fair, Good, Very Good* grading scale that the students are familiar with. The answers and explanations the imaginary actors give often are in the *Good* category and it is up to the students to discuss the best parts of each to come up with a *Very Good* amalgamation.

Topic	Handout A	Solution	Handout B
Programming with algebraic laws	Lookup the proof system for representing binary numbers and write a plain-language description.	A table entry referenced in the homework that describes the two forms of data	Two imaginary students' descriptions: a math major's and a systems course TA's.
Operational Semantics	Rephrase an operational semantics fact to be about code.	Discussion about scope, global variables, and evaluation.	Two imaginary TA explanations, a high-level and a step-by-step, of evaluating an example program in those semantics.
Recursive functions over lists	Complete the first design step doing case analysis on the arguments.	One argument: (`cons` and `nil`) and another: an ordinary S-expression (`nil`, `cons`, or an atom).	Two imaginary students, one argues for case analysis on one argument, one argues for the other.
Higher-order functions: `map`, `foldl`, and `foldr`	Use `map` to write a function.	Code using `map` and anonymous functions in a scheme syntax.	A mathematical definition of "mapping" is presented to describe the code example.
N-Queens with continuations	Implement success and failure continuations with anonymous functions	Anonymous functions that ignore input and return symbols.	A student and TA interaction considers the purpose of these particular continuations if they ignore arguments.
Introduction to ML datatypes	Match three given functions to their types.	The correct function and type pairs.	Given the following function and type pairs, critique these imaginary student descriptions and open questions.
Typing rules for records	For each record given, state whether it should be well-typed.	Two records are well typed and one is not because it has repeated keys.	For each record, critique arguments for why it is and is not well-typed.
Solving conjunction constraints	Given a constraint equation, solve for values of the variables.	Values where the subparts are solvable but their conjunction is not.	Imaginary students discussed how an example constraint is unsolvable even though the parts are solvable.
ML Functors (Modules)	Given code in ML, describe what is tedious or dangerous about it?	Answers that motivate code hiding and preventing the user from passing around intermediate state.	Imaginary students presented comments and open questions about the role of the person using the code.
OOP in Smalltalk	Translate scheme expressions into object method call syntax.	Apply the function as a method to the first argument.	Imaginary students question and comment on translated scheme expressions about the equivalence and semantics differences.
Double dispatch in Smalltalk.	Think of a noun1-verb-noun2 phrase that could be flipped into noun2-more specific verb-noun1 phrase.	Any solution where the more specific verb depends on information about particular choices for noun2	Three examples of noun1-verb-noun2 and for each of those two noun2-more specific-verb-noun1 were given where each noun2 was made more specific as well.

Table 5.1: A high level summary of the first question for each topic as asked in handout A, its correct solution, and a partial solution presented in handout B for critique.

We crafted the Handout B sample answers by distributing important aspects of the handout A solution across the various imaginary actors.

5.6.3 Survey Design

Weekly student and TA surveys assess two related measures: How much participation did the students and TAs *observe* and how prepared students *felt*. Often, we design curriculum based on observations with the hope that our choices affect how students learn. Since there are many steps between recitations and measuring student performance (homework correctness, grades, etc), we decided to track how prepared students felt after recitations as a proxy for learning because it was an actionable, intermediate step. In the short term, we used this information to compare the topics week to week and mentor TAs. In the long term, the data as a whole can help us understand common behavior across topics and semesters.

Observable Participation Both students and TAs reported whether participation was less than, more than, or as expected (a three-point scale) in six categories (the TA version appears in parentheses):

- How much did you (students) ask a question?
- How much did you (students) answer a question?
- How much do you estimate you (students) had a question answered even if it wasn't asked?
- How much did you (students) practice concepts?
- How much did you (students) discuss about concepts?
- How much did you (students) help your (their) peers?

Questions covered three categories because question asking is the primary form of participation in recitations. In addition, questions are easy to observe and can be personally counted. However, we had to rely on self reports because recording the recitation to support video coding would have been outside the service of the course (introducing stress) and requiring TAs to count would have placed too high a burden on them.

We chose the three-point scale to reduce survey fatigue and to fit the categorical analysis we planned. In pre-study testing, using a five-point scale or higher caused irritation when performed multiple times and prompted students to choose answers to finish the survey quickly (all 1s or all 5s) rather than answer earnestly.

Student self-belief in preparation Informally, students and TAs report being more comfortable beginning the assignments before fully understanding what the problem is asking. From that assumption, we wonder if students are more prepared for doing tasks than discussing tasks without our intervention.

If all our homework followed Ren et. al.'s design recipes, then we would ask students how prepared they are for aspects of problem formation like choosing the forms of data of the input

to recursive functions. But since some of our topics require solving other types of problems, we generalized to fit any recitation handout's problem goals. To assess how prepared students felt, they ranked their level of preparation on six tasks related to that week's topic, from most to least prepared. Three of the tasks involved discussing the topic: (1) with a peer, (2) with a TA, or (3) on the class discussion form. The other three tasks were (4) studying the topic with the book and handouts, (5) writing code involving the topic, or (6) debugging programs that used it. We chose these tasks as concrete examples of two distinct types: discussing and doing.

We chose sorting rather than a Likert scale to capture the relative nature of these two types and the subtle differences in the middle ranked tasks. We predicted that the tasks at the extremes would be stable, and did not want a Likert scale to lose information by encouraging students to rank tasks of the two types all the same. If students felt that two tasks were the same rank, they were asked to give them an arbitrary ordering. We believe that if tasks were truely arbitrary then their ranks would even out over many students and many recitations, but if the choices aren't truely arbitrary, making each student order them will show that trend.

Many self-efficacy measures include belief about success and belonging, which we initially modeled by generalizing Enström and Kann's self-efficacy activities for dynamic programming into our topics. But in pre-study testing of the survey instrument, students experienced a lot of survey fatigue sorting more than six items and did not appreciate slowing down to read the specific text for each survey tailored to that recitation's topic. So, we designed the survey to have the same text for the ranks with the place holder "on today's topic" and immediately before performing the ranking they filled in a text field with what today's topic was. The goal of filling in the box was more to remind students of the context of the question and not for an archival purpose for us, although it was helpful in service to the course for tracking how students were interpreting the goal of the recitation. Most students filled in the title of the recitation like in the topics columns of our figures.

Miscellaneous survey design choices Students were graded on participation in the course for completing the surveys each week along with their attendance. Because of normal attendance issues, students were not penalized for missing a small number of surveys. Surveys were due for all students shortly after the last recitation on Friday afternoons. We collected student identifiers to provide them participation credit for the course, but anonymized the identifiers before performing analysis. TAs did not have an external incentive like wages to complete the survey, but the data from the survey was informally covered in the weekly TA training meeting, and so TAs completed the survey for the benefit of those discussions.

The survey allowed for students to submit comments in a free prompt. We hoped that this space would keep students from expressing frustration in the form of the participation rating or preparedness ranking and instead do so in the free prompt. We do not know the extent to which this strategy was successful, but most free text was not topic-specific and was either praise of the TAs or frustration with the length of the recitation material. Most students did not fill out the survey free text.

topic	online						in-person					
	student			TA			student			TA		
	A	B	total	A	B	total	A	B	total	A	B	total
Programming with algebraic laws	54	50	104	10	12	22	42	39	81	9	9	18
Operational Semantics	50	47	97	7	8	15	38	48	86	7	8	15
Recursive functions over lists	55	45	100	7	6	13	37	45	82	7	6	13
Hofs: `map`, `foldl`, and `foldr`	56	45	101	6	7	13	39	45	84	9	5	14
N-Queens with continuations	55	42	97	20	7	27	33	43	76	7	5	12
Introduction to ML datatypes	54	49	103	7	7	14	36	46	82	4	2	6
Typing rules for records	48	43	91	6	7	13	30	37	67	6	2	8
Solving conjunction constraints	52	44	96	7	8	15	37	39	76	4	2	6
ML Functors (Modules)	49	43	92	6	8	14	29	35	64	5	2	7
OOP in Smalltalk	46	42	88	5	4	9	31	41	72	4	2	6
Double dispatch in Smalltalk	49	44	93	3	6	9	30	33	63	3	1	4

Table 5.2: The count of student surveys we received by semester and week.

5.7 Results

5.7.1 Data Summary

Table 5.2 shows how many students and TAs submitted surveys each week in both the online and in-person semesters of the course. We expected fluctuation over time and week-to-week with normal add-drop behavior, variations in student attendance, and survey fatigue.

In designing the metrics from the survey to analyze, we focused on whether participation improved with our intervention, if students were less prepared to discuss topics than to do work on topics without the intervention, and for which topics our intervention might improve preparedness to discuss topics.

5.7.2 Choice of metrics for participation

To compare participation, we summed every response (-1 to 1) to the six participation categories as an overall impression of participation for the recitation session (scale from -6 to 6). Recall the categories were the six 3-point Likert scale questions: had a question, asked a question, had a question answered even if not asked, talked about concepts, practiced concepts, and helped a peer.

We chose this sum as we were not yet interested in the particular categories of participation, but instead wanted to score how participation went overall. The sum is an appropriate metric as we believe the six categories each capture a mostly non-overlapping form of participation.

We also computed an adjusted sum of participation where each response was compared to the mode of that category for each student. For example, if a student reported *less than expected* most often for the *had a question*, then we re-coded their survey responses for that category as 0 for *less than expected* rather than -1 and 1 for both *as expected* and *more than expected*. It follows that in the case that *as expected* was the mode, the survey results would be unchanged. This sum was hoped to be more robust to individual student expectation so we could compare handout A and handout B specifically within the context of the course rather than against fluctuating expectations

of participation based on outside factors.

5.7.3 Choice of metrics for discussion preparedness

To see if discussion tasks ranked below doing tasks, we looked at the median and quartiles of the ranks of each kind of task across each semester. While the survey instrument asked students to visually sort tasks highest up as most prepared and lowest down as least prepared, we transformed the instrument's mapping into a scale of zero to 5 to improve visualization and maintain consistency with bigger numbers being better; a rank of zero was least prepared and a rank of 5 was most prepared. To compare student preparedness to discuss between the handouts, we summed the ranks of the three discussion tasks for each survey. The subtracted difference between the lowest possible sum and the highest possible sum is 10, so we then adjusted the sums to be on a scale of zero to nine. A sum of zero means that the student reported that they were least prepared in all of the discussion tasks in some order and nine means the student was most prepared in all of the discussion tasks in some order. The sum is an appropriate metric for comparing preparedness to discuss between handout A and handout B because each task captures a different method in which students discuss the topic, and we are not yet interested in the particular tasks in this study.

5.7.4 Statistical analysis method choice

To determine if there is a difference in the distributions underlying the participation sum and discussion preparedness sum between handout A and handout B, we chose the Mann-Whitney U (MWU) [MW47] test. MWU is an appropriate choice because it works for ordinal data since it does ordinal comparisons ($<$, $=$, and $>$) on each pair when computing the U statistics. Our sums are on small scales and are discrete, making them behave like ordinal data. We are able to run MWU for three alternative hypotheses: whether handout A and handout B's distributions are different (called a two-sided test), whether handout A's distribution is less than handout B's, or whether handout A's distribution is more than handout B. Being able to choose the alternative hypothesis other than just difference benefits our study since, if there is a difference, we care if the difference has a direction: whether handout A or handout B produced larger sums; it could be the case that their distributions are different, but one does not produce larger sums than the other. We interpret a larger sum as better.

For each test, MWU computes our alternative hypothesis's U statistic: the sum of a comparison function for each metric from handout A with each metric from handout B. The null hypothesis for each test is the negation of each of the alternative hypotheses' ordinal comparison operators: inequality negates to equality, less than negates to equal or greater than, greater negates to equal or less than. MWU computes the p-value by comparing our alternative hypothesis' U statistic with the U statistic for the null hypothesis: a known value approximated using a normal distribution and our sample size. From the literature on the application of p-values, we should not use the p-value to decide conclusively for which topics the interventions did or did not perform [And19], but we do use the p-value to order the topics by interest and prioritize which topics warrant further study with our intervention.

topic	online				in-person			
	student		TA		student		TA	
	U	p value	U	p value	U	p value	U	p value
Programming with algebraic laws	1382.0	0.835	77.0	0.252	875.5	0.59	25.5	0.183
Operational Semantics	1394.0	0.109	26.5	0.906	931.5	0.866	34.5	0.48
Recursive functions over lists	1276.5	0.787	25.0	0.609	794.0	0.719	25.5	0.562
Hofs: `map`, `foldl`, and `foldr`	1039.5	0.125	24.0	0.703	855.5	0.845	16.5	0.45
N-Queens with continuations	1136.0	0.891	47.5	0.217	667.0	0.654	18.0	1.0
Introduction to ML datatypes	1286.0	0.807	23.0	0.894	857.5	0.783	6.5	0.34
Typing rules for records	872.5	0.197	23.5	0.772	623.5	0.383	7.0	0.864
Solving conjunction constraints	1103.0	0.759	17.5	0.242	641.5	0.402	4.0	1.0
ML Functors (Modules)	935.5	0.34	14.0	0.209	410.0	0.182	3.5	0.693
OOP in Smalltalk	878.0	0.441	10.0	1.0	557.0	0.358	7.5	0.153
Double dispatch in Smalltalk	1099.5	0.867	12.5	0.429	487.5	0.921	N/A	N/A

Table 5.3: The U statistic and p value determining handout A is **not** likely to produce participation sums different than handout B using the Mann-Whitney U test from student and TA survey results. There were not enough surveys in the last in-person week from TAs to run the analysis.

To meet the pre-conditions of MWU, we only analyze per topic. MWU needs data samples to be independent and analyzing surveys from the same student in multiple weeks breaks independence.

As our implementation of MWU, we use the pingouin python library [Val18], which is a statistics library built on `scipy.stats` [JOP+–]. In addition to computing the MWU statistic and p-value, pingouin also checks that the variance in the data is acceptable and computes a common language effect size (CLES) [VD00], which can be thought of as the percentage of pairs from the two data sets (handout A and handout B) that support the outcome. We use the CLES to measure the effectiveness of our intervention.

5.7.5 Participation

As shown in table 5.3, we found no evidence to reject the null hypothesis in the raw participation sum for students or TAs. Many factors impact participation: TA teaching efficacy, cohort effect, and individual student expectations. While it is hard to control for all of those in a study over the whole semester, considering these factors led us to design the section participation metric of the adjusted sum.

We ran MWU on the adjusted participation sums also using the two-sided test to determine which topics might have been more impacted by the intervention. I.e. to determine which topics we can reject the null hypothesis that handout A produced sums equal to handout B for particular semester and topics. Table 5.4 shows that given our data we can reject the null hypothesis that handout A and handout B produce equal participation in the online semester with weak evidence ($p < 0.1$) for the topic of operational semantics. We can reject the null hypothesis in the in-person semester with evidence ($p < .05$) in the topics of designing recursive functions over lists and implementing N-Queens using continuation passing style (CPS) We ran MWU again on each of these topics and semesters for the less and greater alternative hypotheses to determine direction

topic	online		in-person	
	U	p value	U	p value
Programming with algebraic laws	1382.0	0.834	837.0	0.864
Operational Semantics	**1411.0**	**0.0827**	750.5	0.154
Recursive functions over lists	1394.5	0.266	**647.5**	**0.0753**
Hofs: `map`, `foldl`, and `foldr`	1177.0	0.562	837.5	0.716
N-Queens with continuations	1187.0	0.815	**539.5**	**0.0665**
Introduction to ML datatypes	1245.5	0.599	848.5	0.847
Typing rules for records	900.0	0.278	596.5	0.598
Solving conjunction constraints	1115.5	0.831	783.5	0.505
ML Functors (Modules)	965.5	0.479	514.5	0.928
OOP in Smalltalk	993.0	0.815	503.0	0.121
Double dispatch in Smalltalk	1191.0	0.365	493.5	0.989

topic	semester	direction	U	p value	CLES
Operational Semantics	online	A > B	1411.0	0.0413	0.6
Recursive functions over lists	in-person	B > A	647.5	0.0377	0.611
N-Queens with continuations	in-person	B > A	539.5	0.0332	0.62

Table 5.4: MWU run on the adjusted participation sum. We found interest in results with ($p < 0.1$), and we re-ran MWU with the one-sided alternative hypotheses to determine direction and CLES. For one topic in the online semester, handout A had more participation. For two topics in the in-person semester handout B had more participation. The CLES shows that a randomly sampled survey using the greater handout has higher participation than a randomly sampled survey from the other handout at least 60% of the time.

and CLES. If the p-value went up for both of these tests, then we would not be able to conclude a direction. In all three cases, there was a one-sided test where the p-value went down, and we found that in the online semester, given the data, we could consider the alternative hypothesis that handout A produced more participation than handout B for the topic of operational semantics. In the in-person semester, we could accept the alternative hypothesis that handout B produced more participation than handout A for the topics of recursive function design on lists and N-Queens with CPS.

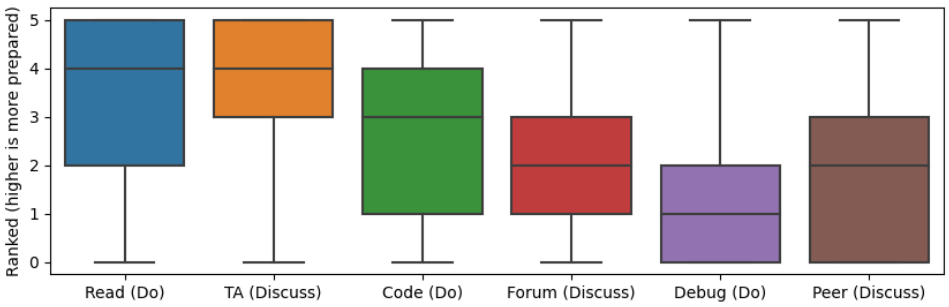
In doing these one-sided tests, we computed the common language effect size and found that at least 60% of all the handout A handout B pairs would support these effects.

We could not compute this value for TAs since we did not take TA identifiers from their surveys to keep maximum anonymity (and least stress) for TAs in service of the course; without identifies, we could not compute the mode between surveys for this adjusted sum.

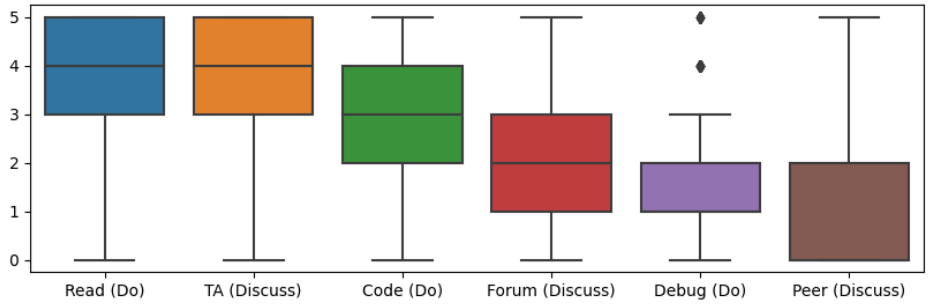
5.7.6 Task Preparedness Ranking

To understand whether improving how prepared students were to discuss topics was important, we wanted to know how discussion tasks compared to doing tasks for students without the intervention.

Figure 5.1 shows the rank percentile and median average for each task from students using handout A for the online and in-person semesters. Preparedness to look up answers in the book and handouts and to discuss topics with TAs ranks highest. These are the tasks students had just



Online Semester



In-person Semester

Figure 5.1: For just handout A (No intervention), the rank averages and percentiles for each task over each semester

preformed to do the recitation.

Preparedness to debug and test their projects ranked lowest, as is reasonably expected since that step is often done last; however, we do ask students to test and debug code incrementally and throughout the homework. In comparing the median lines of the box chart, we see that preparedness to discuss on the class forum and with a peer are roughly the same, with preparedness to discuss with a peer having a wider coverage.

The chart shows that students are more prepared to write code after recitation than to discuss concepts on the forum or with a peer. The existence of a difference reassures us that improving the relative rank of student preparedness for discussion tasks could positively impact the original issues in TA office hours: students only asking for help after jumping into code without sufficiently understanding concepts. In contrast, if the discussion tasks had instead ranked highest for students without the intervention, then we would most likely be solving the wrong problem and further investigation on the correlation between student preparedness to discuss topics and TA office hour interactions for specific tasks would not be fruitful or worth more disruptive methodologies (like TA and student journaling).

Also using MWU, we compared the sum of the ranks of the three discussion tasks: discuss with a peer, discuss with a TA, and discuss on the class forum. Table 5.5 shows that the rank sums were different using handout A than handout B in 5 topics with weak ($p < 0.1$) and regular ($p < 0.05$) evidence. This evidence made us hopeful that we could see a pattern between the topics impacted. When testing these topics and semesters with MWU again using the other alternative hypotheses, we found the direction of handout B ranked discussion tasks higher than handout A in all 5 topics. All five one-sided tests reported a CLES of at least 0.6. With similar analysis as with

topic	online		in-person	
	U	p value	U	p value
Programming with algebraic laws	1316.5	0.829	801.5	0.871
Operational Semantics	1024.0	0.272	**660.0**	**0.0267**
Recursive functions over lists	1128.5	0.447	823.0	0.933
Hofs: `map`, `foldl`, and `foldr`	1161.5	0.498	761.0	0.291
N-Queens with continuations	998.0	0.248	**532.0**	**0.0607**
Introduction to ML datatypes	1178.5	0.338	794.5	0.755
Typing rules for records	1004.0	0.825	461.5	0.23
Solving conjunction constraints	1110.5	0.806	**564.0**	**0.0987**
ML Functors (Modules)	1028.0	0.843	392.5	0.115
OOP in Smalltalk	789.5	0.135	**452.0**	**0.0345**
Double dispatch in Smalltalk	**848.5**	**0.0746**	411.0	0.246

topic	semester	direction	U	p value	CLES
Operational Semantics	in-person	B > A	660.0	0.0133	0.638
N-Queens with continuations	in-person	B > A	532.0	0.0303	0.625
Solving conjunction constraints	in-person	B > A	564.0	0.0493	0.609
OOP in Smalltalk	in-person	B > A	452.0	0.0172	0.644
Double dispatch in Smalltalk	online	B > A	848.5	0.0373	0.606

Table 5.5: Using MWU with a two-sided tail, student reported discussion tasks using handout B rank higher or lower than handout A in five topics, one in the online semester and four in the in-person semester. When checking the tails, handout B ranked discussion tasks higher than handout A with CLES > 0.6, showing that when pairing samples from handout A with handout B per topic at least 60% ranked discussion higher with handout B.

the adjusted preparedness: at least 60% of pairs of surveys from handout A and handout B for each of these 5 topics would have handout B ranking discussion preparedness higher than handout A.

5.8 Discussion

This teaching intervention provided modest framing agency towards the end state of the initial question of the recitation activity. Specifically, students were given a known initial state (the lecture materials) and a known set of logical operators (critique an existing answer in the form of a playscript). Then, students were asked to determine the end state (discussion of their critiques). The instructors intended for students to exhibit the epistemic agency for learning that occurs when TAs solicit participation by first modeling their own understanding and then asking students to compare and contrast with their own understanding. The choices students were expected to make would be in what features of the existing answers they would choose to agree or disagree with, as well as to what degree they believed the answers are good or could be better and why. From the survey results, we do not know the epistemic agency students actually exhibited during these activities and more focused studies should do analysis on student discussions to test the relationship between the framing agency and the epistemic agency assumed by the intervention. Since TAs reported success in facilitating the discussions, students are assumed to have chosen some aspect as important since no suggested aspect was given.

5.8.1 Interpretation of the results

Our primary goal in interpreting the results is to highlight potential places where our intervention may have helped students participate in recitation or prepare to discuss topics in office hours. The biggest consideration in interpreting the results is that there are a lot of variables and that the data is self reported via surveys. We keep these facts in mind and do not claim that our intervention predicts improved observable metrics of learning. Instead we hope the work gives us insight to design future studies in which we measure more definitive, but also more disruptive, measures of student learning. By focusing our efforts in this way, we seek to gain insights without significantly increasing stress and work loads for students or TAs.

TA interpretations with informal interviews While not coded nor included as formal results, we interviewed four TAs who had taught in both the online and in-person semesters of the course in an effort to consider our interpretation of the results in the context of the impressions of the relevant TAs. These interview responses should not be considered as data for the effectiveness of the intervention; however, the comments might provide context from a perspective other than curriculum and study design. In the following paragraphs discussing our insights, we add commentary from these TA interviews.

Participation We do not believe our intervention significantly impacted the perception of participation. From the original participation sums, we believe explanations for the lack of impact could be that (1) our intervention was too small or (2) it was lost in the noise of other factors that

affect participation such as TA teaching ability, cohort familiarity, or individual student expectations. Even when controlling for individual student expectations with the adjusted sum, the MWU two-sided test showed only weak evidence that we could reject the null hypothesis that handout A and handout B were the same in three topics. Consistent with this weak finding, we note that in the interviews, the TAs did not observe a difference in which type of handout consistently showed improved participation.

We consider the problem goals using handout B of the three topics with weak evidence using table 5.1. The Operational semantics handout models rephrasing a fact about math into a fact about code. The Recursive function handout models the data formation step of a design recipe. The N-Queens handout models the algorithmic purpose of code that may appear to be ill-designed, but actually fits a form of data when considering a function's arguments as part of its form. In answering our first research question, we do not see striking common problem goals between all three of these, and so we conclude there is not sufficient evidence our intervention improves participation.

In commenting on participation in their interviews, the four TAs focused on the willingness of students to talk about each week's topic and how they believe that willingness was proportional to the amount the students were already talking, which we would consider a cohort effect. "One [handout B recitation section] was a lot more vocal and participatory [than another handout B section], and would always say something as soon as we came out of [small groups for answering the first question], unprompted." (In this quote, internally understood terms were replaced with what they refer to in relation to the study). Controlling for cohort effect is difficult in a class where students self-select into sections based on availability.

Discussion Preparedness Our intervention had the most positive impact on the topics where handout B embedded *why* the imaginary student is giving the answer. In four of the five topics, the purpose of the first question was to build understanding of a new computational or algorithmic concept. Using table 5.1, we consider the handout A versions of each of the questions. The Operational Semantics handout asks students to reason about a mathematical notation in terms of something they already know – code. The N-Queens handout asks students to write code they have seen for a new algorithmic purpose. The Solving Conjunction Constraints handout asks students to build intuition for the algorithm that underlies type inference using a programming language they learned the prior week – ML. The Double Dispatch in Smalltalk handout asks students to build intuition for what the algorithm is doing – adding more subclass specific information – when it makes subsequent method calls. The fifth topic, the Introduction to OOP handout, asked students to translate code from one syntactic style to another. In turning the question into a critique for handout B, we added information about the semantic differences in the student explanation which unintentionally motivated *why* we do OOP. In the first four of these instances, the handout A question did not have the problem goal of producing code using the design recipe, but instead understanding an algorithm, and in the fifth instance our handout B addressed understanding of the underlying semantics. As a result, we tentatively conclude that the investigating critique approach could be best applied to PL topics that need motivation for *why* they are being learned,

specifically when it is not clear what problem they solve. For the topics not listed here, we speculate that the introduction of syntactic forms to represent the language features introduced helps provide concrete motivation to their usefulness and application.

We found this result counter-intuitive, as we anticipated our intervention would help the recitations that introduce new syntax and semantics more than the ones that introduced algorithmic ideas. In a post-study interview with a TA that taught from both handout A and handout B, their impression was that in both cases follow-up questions had to do with what the student saw in the handouts. In the case of handout A, it was their answers and in the case of handout B it was in the text of example solutions. "[B handout] students would ask more rigid questions skewed more towards syntax. [A handout] students would ask more about the underlying thought because they didn't realize they had gotten it wrong at first." The TA's impression and the data together contrast priorities of curriculum design and teaching on the usefulness of the intervention since TAs found it useful as a reference for new language feature syntax, but we found it useful for improving discussion preparedness on algorithmic concepts.

We did not find any cases where students were less prepared to discuss topics using handout B. If there were such cases, our intervention would potentially have the opposite of its intended effect: increasing the gap between a student's ability to write code and discuss course concepts on the forum or with a peer. Since we believe our intervention is at worst neutral, we recommend using the handout B approach on a case-by-case basis whenever the topic might benefit from critique rather than as a rule for the beginning of every recitation. To decide when to apply the technique, we would prioritize problem goals that benefit from answering *why* we are learning the topic, which we see when problem goals for the topic front load theory. Enström and Kann's work supports this strategy: they attribute a potential source of difficulty with one of their topics, complexity analysis and Big O notation, to a lack of motivation and perceived usefulness early on.

Visual metaphors The other interesting commonality to the topics for which handout B had the most positive impact is that understanding them often relies on TAs and the lecture videos inventing visual metaphors rather than modeling code. Operational Semantics structures programs as trees. The lecture videos on continuations structure each recursive call as a box and the continuations as wires that compose like circuit-diagrams. Solving conjunction constraints did not have an official visual metaphor, but in an interview a TA shared their experience improvising one for it: "One time while explaining, I used a Venn Diagram and people [stopped and] looked at me and then I actively started speaking louder so other people could pay attention. Often I was 'half-teaching' to those that were listening.". As solving conjunction constraints is the algorithm underlying the homework lesson on implementing type inference, the visual metaphor for potential concrete types a type variable could have was well suited for Venn Diagrams. OOP and Double Dispatch both rely on understanding message passing, which visually we represent with circles or other shapes (for objects) and labeled arrows (method calls) as a pseudo-declarative view of the call stack. While we do not know if our intervention specifically helped TAs to build these metaphors, giving students the time to explain examples might result in these drawings regardless if the student answers are correct or not. We believe that with corrective feedback, if the student

answers are correct, then TAs do not do this visual modeling and the students lose out on learning a useful communication tool.

Online vs in-person We were surprised that our intervention showed the most effect on student discussion preparedness in the in-person rather than in the online version of the course. We thought the fact that TAs and students followed the handout as written more in the online recitations would make the difference more significant in that version. We want to know whether our assumption was incorrect or whether students in the online setting had more resistance to changing how prepared they felt. In other words, in the online setting, were there more dominating factors that kept preparedness stable?

In the post-study interview, TAs reported lower barriers for students asking clarifying questions and following up in the in-person recitations, which TAs took advantage of. All four TAs mentioned they off-loaded the ideas and knowledge that benefited the entire group to shared tools like whiteboards in the in-person semester. The offloading of ideas to whiteboards supports our idea that an aspect of our intervention worked well with topics that benefit from visual metaphors. Lastly, in the in-person semester, the groups were in the same physical space when discussing the first question, but were in breakout rooms when online. Studying the interactions during the group solving of the problem and not just when they reconvene with TAs might provide insight into the potential difference and into designing interactions better suited for online learning.

5.8.2 Threats to validity

To prevent survey fatigue, we made design decisions to let the student submit the survey as quickly as possible. However, to combat uncritically turning in the survey, students had to activate (touch or click) the slider for each of the participation categories and the task sorting widget even if they were leaving it unchanged. When reporting participation, they could leave the sorted rankings as they were. In looking at the box plot in figure 5.1, we present the tasks in the order they were presented to students and see that the 50th percentile follows the trend of the students not changing the order.

To check whether these decisions impacted the validity of our analysis, we calculated the number of surveys that did not change the order of responses: 8% for the online semester and 14% for the in-person semester. While non-trivial, we believe our analysis was conservative enough that the risk is more in not discovering areas where the intervention could have had an effect, rather than in false positives. Also, since this behavior would impact students using handouts A and B similarly, the impact on comparing the same tasks between groups, as we did, is affected less than comparing between tasks, like comparing doing vs discussing trends.

Follow up studies could control for this confounding variable by randomizing the order of the tasks and tracking student engagement as metadata to ensure students are engaging with the instrument. Since our students were taking the same survey 11 times, we did not want to randomize the order. We anticipated more students would leave the tasks ranked in their original order if they had to reread it carefully each time. Future work could focus on topics that seem receptive to our intervention, subjecting students to fewer but more comprehensive surveys.

5.8.3 Future work

Further analysis on the data In further investigation using the data we collected, we want to focus on particular preparedness tasks, specifically preparedness to discuss with a peer, discuss on the class forum, and writing code, and also on each participation category. Our initial analysis in this study provided us an overview, but in using the sums we threw away information about differences between specific categories and tasks. In focusing on particular categories and tasks, we might be able to understand the impact of our teaching interventions on any particular aspects that are not as resistant to intervention.

Further interesting questions As mentioned, we would focus future work on improving our PL course starting with the five topics highlighted in our results on student discussion preparedness. We would want to compare critique activities like the one we did for this study with ones that take advantage of the visual metaphors of the algorithms underlying these topics. In doing so, we might find a less bespoke way of preparing TAs to hold the first discussion of recitation.

Measuring participation In taking survey measures, we found that surveying students multiple times over the semester caused fatigue despite our best efforts and our ability to keep the survey time short (roughly less than two minutes). In modeling the survey this way, we did get submissions from all students and gained insight that was useful. We benefited in the same way as we had when having students write a thought or two on a note-card when leaving class in service to the course (not for research study). Although the note-card method was more time consuming and less regular, TAs perceive these formal surveys as more disruptive and annoying.

In future work, passive observations like having a third party observer count instances of participation would be better as a primary metric, with periodic surveys to contrast. We anticipate this approach would be easier to implement in-person where having a guest in the room is less disruptive than having an extra caller in a video session.

Measuring Preparedness In a future study focused on particular topics, which would limit the scope to a few select weeks, topic-specific and problem-goal-specific measurements become easier. In choosing to rank items rather than use a Likert scale, we successfully avoided the issues of close choices being indistinguishable, but instead encountered a significant issue of students not engaging with the instrument and leaving the sorting as presented. In a more focused study with fewer surveys, we would consider having students allocate some portion of 100 points into each of the tasks and deriving weighted ranks to make students engage with the instrument as well as gain weighted information about the ranks. Although this structure would introduce the ability to rank two tasks the same by allocating the same points to them, the likelihood of that issue compared to a Likert rating would be much lower, and when it happens, much more intentional.

In studies on self-efficacy, researchers code language in work journals students keep. In our course, adding work for students is not viable, but in future work we want to measure the reality of their self-efficacy, whether through metadata on their interactions with TAs, the class forum, or on homework submission. In service to our inciting problem and for the specific topics

we have identified in this study, we would measure how student self-belief correlates with higher quality TA-student interactions during office hours.

5.8.4 Conclusion

Our intervention opened 11 recitations with critique-style questions that model syntax and problem goals to focus student attention on concepts. We hoped to see improvement to participation because of a shift to explanatory feedback to everyone rather than corrective feedback for the first correct answer. We hoped to see improvement to how prepared students were to discuss concepts relative to doing the homework since we believe that students begin writing code before understanding the concepts leading to lower quality TA-student interactions. Our intervention does not seem to impact an aggregate measure of participation reported on student and TA surveys. Our intervention may impact how prepared students are to discuss concepts on a subset of topics. We believe the common features of the topics our intervention positively impacted were (1) an emphasis on understanding algorithmic ideas that need more motivation to answer *why* we are teaching them and (2) a reliance on visual metaphor for understanding rather than (or in addition to) exemplary code. Our intervention had the most effect on in-person recitations, which suggests two future directions: (1) designing different interventions that benefit the specific challenges of online recitations, such as the lack of open group conversation and (2) considering in more depth whether features of this study's intervention could help students reach out for help earlier.

6

Student agency via self-regulated learning in a DSL design seminar

Upper-level special-topics seminars enable Computer Science (CS) educators to share an in-depth area of study with undergraduate and graduate students alike. Undergraduate students discover interests beyond the core curriculum in preparation for portfolio-building endeavours like internships and end-of-degree capstone projects [Ria20]. Graduate students explore a new research area or practice deeply in their own. For both types of student, seminars can facilitate active- and deep-learning outcomes – inviting students to "engage in their learning processes"[Pri04] and apply their learning to problems of their own devising [DKI08].

6.1 Our course

We taught an upper-level programming-language seminar in which students learned to read and discuss research papers on Domain-Specific Languages (DSL) and designed, implemented, and presented their own prototype DSLs. A DSL is a programming language with abstractions, tooling, and runtime support for a particular application domain [vDKV00]. Popular examples include LaTeX for document writing, SQL for relational database queries, and shader languages like HLSL or GLSL for graphics applications. The papers students studied introduced a variety of DSLs and described how to design, implement, use, and evaluate DSLs. To apply what they learned in the reading, students implemented their own DSL designs using a meta-programming tool chain described in recorded lectures and assigned papers, wrote technical papers in the style of the DSL papers they read, and peer-reviewed each others' work.

The class started with 30 students, falling to 21 because of typical add-drop behavior. Despite making A-level progress, two students dropped late in the term. Ultimately 19 students finished the course; three were programming languages (PL) PhD students, 5 were non-PL masters students, and 11 were junior and senior undergraduates.

6.2 The problem

A typical seminar-style course with a final project would not address the wide disparity in our students' knowledge levels, nor the breadth of skills they would need to meet their own goals. In addition, we encouraged students to choose domains that interested them, but domains differ greatly in their difficulty levels and in our ability to facilitate work in them. To summarize, the challenge of the course was to maintain fairness and rigor while allowing for student agency among all dimensions of the project.

6.3 Our contributions

To accommodate this breadth of experience and content, we integrated several popular teaching methods: a Blended Learning environment [Hra19] with online materials and homework, in-class discussions focused on Peer Instruction [SWA+09] and collaboration roles [BC09], and a Just-in-time-Teaching (JiTT) [GWMB04] approach in which we used students' in-class notes and feedback to improve upcoming online materials and in which results from homework assignments guided in-class discussions in turn. Given that framework for the course, in this paper we focus on just two teaching interventions: (1) a novel assessment rubric that follows Epistemology for Science principles[Rus14] and (2) an accompanying grading strategy that enables the goal-setting and reflection phases from the Self-Regulated Learning (SRL) literature [Zim08]. In the following sections, we describe why we chose these two interventions, how we implemented them, what outcomes we observed in student work and behavior, and how we interpret the impact of those outcomes.

6.4 Study-specific Background

6.4.1 Student regulated learning

Students needed to be able to implement the ideas from the reading in their projects. We wanted the assessments in the course rubric to guide them in that process. Since student project variation makes each implementation different, we sought teaching strategies that enabled student agency in planning their projects, such as the strategies found in Self-Regulated Learning (SRL) [Zim08]. SRL describes strategies where students are "metacognitively, motivationally, and behaviorally active participants in their own learning process"[ZP86]. When students are taught using SRL strategies, they display greater skill and self-efficacy [Zim08]. Increases in self-efficacy – "judgments about how well one can organize and execute courses of action" [BS81]– have been shown to be predictive of increased academic performance [BNA15].

In the literature, SRL techniques have been implemented and evaluated most often in introductory and intermediary CS courses with successful outcomes in adding "Forethought" and "Self-Reflection" phases [FVF14]. We anticipated that analogous outcomes at the seminar level would enable student agency by helping students plan the ideas that their current project iteration would and would not exhibit, iterate on their project, and then reflect in the materials they submitted on how their work showed learning.

6.4.2 Fostering participation

As introduced, our investigation reflects the choices of other theory heavy, computer science curricula and appears in Enström and Kann's study of the trickiest topics in their algorithms course [EK17]. In their approach to improving in-lecture participation, they tested mechanisms like raising colored cards, pointing to sides of the room representing multiple choice/true or false, and anonymous clickers [LGP13] to lower the barrier for students to answer questions with successful results. While our recitations are already interactive with prepared questions, we saw an opportunity to build on their insights on the importance of how the questions are crafted to reduce cognitive load. Specifically, Enström and Kann say "Preparing good clicker questions, even in the absence of clickers and even in the absence of response cards, is a practice that we recommend." We observed similar issues to those they encountered when prompting students for participation such that the TA implies a correct order to the answers or to partial correctness in general which crowds out questions or insights that do not fit to answering the question. This crowding out introduces cognitive load of navigating the form of the prompt in addition to answering it, which prompted us to investigate feedback other than corrective feedback.

6.4.3 Knowledge construction between students and instructors

6.4.3.1 Do student's believe they can ask questions?

We use a similar definition of self-efficacy as Enström and Kann: confidence that is timely and applicable to the content or topic, focused on a concrete task or action the student can perform, and not inherent to properties of the student. This self-efficacy can change or be changed over time and has been measured as common properties to survey questions defined in the original paper on self-efficacy in CS by Ramalingam and Wiedenbeck [RW98]. From their work, we designed our survey instruments to measure self-efficacy around how prepared students were to discuss topics and to see if we would find similar results positively correlated with self-efficacy as outlined in detail in Enström and Kann's related work and as a major component of literature on predicting a CS student's performance [HIP+18].

6.4.3.2 What can we change?

In our study, we have control over the TA instruction and handout materials for recitation. Toma and Vahrenhold [TV18] encountered similar concerns in their algorithms labs about student self-efficacy and emotional factors during TA student lab interactions, especially at scale. While their study focuses on the emotional and belonging side of self-efficacy, their approach in assessing the usefulness of the lab materials and through the lens of TA student interaction quality reflects our problem and student population. Their study show a relationship between intrinsic motivation and cognitive load with help seeking behaviors, and how these relationships vary by topics over time. We designed our survey instruments inspired by measuring preparedness to engage in help seeking behaviors by topic, but chose task preparedness using Enström and Kann's interpretation of self-efficacy rather than this belonging oriented one after feedback on an early instrument design caused confusion and frustration at the believed unchanging quality in our high stress course. For

example, Toma and Vahrenhold's question "Considering the difficulty of this course, the teacher, and my skills, I think I will do well in this course." was not perceived as being topic oriented. As a result, we chose the language of *prepared to discuss* and *prepared to do* which measures a self-belief, but with student agency and potential to change over the semester.

6.4.4 Where can we help TAs?

In other aspects of CS, TAs spend much of their time in setting up the focus of the problem. In Ren, et. al.'s study of students and TA interactions, they used a design recipe for designing functions to measure and categorize where in the problem TAs helped students [RKF19]. A design recipe categorizes what high level step a student is doing during function design and implementation. They found most of the time was spent in the data/problem formation step and the implement the function body step. In our PL course, we follow a comparable TA and student design recipe for many of our problems that involve designing recursive functions, and we observe a similar effect in recitation TA interactions. However, many of our problems involve digesting a high level theory concept that would further burden the first design step, so this work helped us decide to focus on problem formation and not the entire recitation handout.

6.5 Course Design

Students built embedded domain-specific languages using Haskell as the programming environment and Template Haskell [SJ02] as the meta-programming environment. To "build the right thing," students discovered language designs, implementation details, and usability/evaluation criteria that DSL authors presented in research papers, compared these in class throughout the semester, and authored criteria to evaluate their own DSLs and peer review each others'. To "build the thing right," we facilitated in-class lab activities where students inquired [JRR+09] into exemplar artifacts – toy implementations of a DSL that illustrated the implementation techniques.

6.5.1 Assessment and grading

We anticipated difficulty in distinctly categorizing the students' actions and thinking. If we used Bloom's taxonomy as our abstraction for lesson planning, then we would also need an unambigu-

	SE&D	FP	A&R	TRC	TW&R	D&U	CG&M	LFED	CF&E	MR&PG
Req	00-1	00-2	00-1	01-2	00-1	00-1	01-2	01-2	01-2	04-1
Recall	01-1	01-1	01-1	03-1	05-1	05-1	03-1	03-1	11-1	11-1
Recog	10-1	23-1	05-1	07-1	07-1	07-1	07-1	07-1	07-1	15-1
Reason	16-1	23-1	12-1	15-1	24-1	12-1	24-1	23-1	13-2	16-1
Radiant	23-1	23-1	23-1	15-1	24-1		24-1	23-1		

Figure 6.1: An example student matrix of rubric items (learning objectives as columns, levels of understanding as rows) with cells representing the assignment on which the student demonstrated proficiency shown in the format $nn - m$, where nn is the class number and m is the assignment number.

ous rubric to communicate expectations to students. Rather than maintaining one representation for assessment and a second for grading, we wanted a single abstraction. When designing this single abstraction, we wanted to preserve the benefits of the two dimensions of the revised Bloom's taxonomy but with course-content-specific and easily observable dimensions.

We chose 10 learning objectives that rewarded students for incorporating the breadth of tasks from the readings into their projects. In our classroom environment, we anticipated 5 levels of understanding where students would show evidence of learning with increased demands as an active participant in each objective. Students would communicate what they learned by claiming which items from this 10 by 5 matrix their work exhibits and why.

To foster goal setting, we encouraged students to adapt project assignments to better suit their needs. Specifically, students altered assignments to (1) earn a subset of the items the assignment prompt suggested with the intention of earning them later, (2) earn objectives previously eschewed, (3) earn objectives in advance of the suggested items, and (4) skip already-earned objectives. Student work became a loop of setting rubric items as goals, iterating on the project as guided by suggested or modified homework instructions, and then reflecting on what their work did/did not show to claim credit for specific rubric items. We presented our assessment plan to students as a single, course-wide rubric rather than grading each assignment with its own rubric. Feedback given to students before each loop iteration took the form of either accepting their claim of learning and its accompanying evidence or acknowledging the claim but rejecting the evidence and telling them how to improve the project to try again.

Ten learning objectives We chose the following 10 learning objectives because they defined the space that all student work in the course would share along various dimensions:

1. Software Ecosystem and Development (SE&D)
2. Functional Programming (FP)
3. Abstractions and Representations (A&R)
4. Technical Reading Comprehension (TRC)
5. Technical Writing and Rhetoric (TW&R)
6. Documentation and Usability (D&U)
7. CodeGen and Metaprogramming (CG&M)
8. Language FrontEnd Design (LFED)
9. Composing Feedback and Evaluation (CF&E)
10. Metacognition Reflection, and Personal Growth (MR&PG)

Objectives 1 through 3 represent the technical implementation skills that students would get by completing the DSL project and "building the thing right" by making appropriate implementation

choices. Objectives 4 and 5 represent the research-oriented skills that would familiarize undergraduate and graduate students with DSL research. Objectives 6 through 8 represent design skills that ensure students "build the right thing" from multiple perspectives: the domain expert, the DSL library programmer, and the DSL end user. Objectives 9 and 10 represent two human-interaction skills for understanding DSLs in a real-world context. By structuring assessment around these objectives rather than per-assignment completeness, we rewarded students for exploring the breadth of DSL design and implementation but did not prevent them from going deep on project-specific paths that interested them.

Five levels of understanding We rewarded five distinct levels of understanding for each learning objective that we termed: *Requisite*, *Recall*, *Recognition*, *Reasoning*, and *Radiant*. *Requisite* knowledge means being able to follow instructions and answer questions where all the evidence was in the homework materials in a fashion similar to a multiple-choice question accompanying lecture material. *Recall* knowledge means being able to reference or cite an idea from a previous homework or class discussion in support of an answer, like answering a fill-in-the-blank question with a vocabulary word from a previous lecture. *Recognition* means being able to see a concept from an old question in a new one and apply a similar approach, like answering a word problem that needs to be restructured in the form of a question. *Reasoning* means being able to reduce real-world problems that are not made for the course into toy problems similar to ones seen in the course and then apply the answer to them, like declaring all the assumptions about a topic before beginning a short essay. *Radiant* means applying (or describing how to apply) the solution and answers discovered in the course to real-world problems. This level means addressing the assumptions made at the previous level and being concrete and actionable in the execution.

To claim a homework submission showed learning at a particular level, students had to describe where in their homework submission evidence for the claim existed and briefly discuss the knowledge they needed to produce that evidence. In these discussions, simply describing where the knowledge came from and what they had to do to apply the knowledge showed whether the evidence satisfied the level of understanding. For example, knowledge from a previous assignment's in-class discussion modified to tackle a new problem is enough detail to demonstrate the *Recall* level of understanding.

Semester-long rubric Rather than assess students with a bespoke rubric for each iteration on their project, we expected students to exhibit learning in each learning objective and each level of understanding at least once at some point in the semester. We call a learning objective at a particular level of understanding a rubric item. We designed the *Requisite*, *Recall*, and *Recognition* levels to be earned quickly since we knew students would (1) come in with knowledge from their introduction to PL and software engineering courses; (2) apply the reading and pre-recorded lecture material for the first time immediately on the homework; and (3) see patterns of the same skills being used in further reading and iteration. Rubric items for these first three levels of understanding were often accompanied with descriptions in the form of "students will be able to..." (SWBATs); later SWBATs became more abstract as we expected students to become more inde-

pendent in choosing what their work would show and how.

Learning at the *Requisite* and *Recall* levels readily apply to any CS course, including introductory ones, where students solve problems in the intended way. We believe learning at the *Recognition* level particularly applies to project courses where students apply earlier knowledge to new problems of their own design. By rewarding learning at the *Recognition* level specifically, we anticipated that students would bridge the gap between the class environment – the skillset from lectures, toy examples, and the reading – with the knowledge needed in their self-selected domains.

While exploring their domains, students would quickly find design problems that go beyond the scope of the technical content taught in the course. The highest two levels of understanding, *Reasoning* and *Radiant*, focused on inter-operating between classroom and real-world DSL problems. We hoped that learning at these levels would enable students to follow an *Epistomology for Science* approach and discover good DSL design lessons amidst implementation details from both the readings and their domains.

Communicating grades Over the entire semester, we maintained one assessment file for each student in their Github repository. The assessment file contained a 10x5 matrix of the rubric items. Learning objectives formed the columns and levels of understanding formed the rows. As students demonstrated proficiency in each rubric item, we filled in the intersecting cell with the contributing homework assignment and left yet-to-be earned proficiency levels blank. To illustrate how students saw their progress, Figure 6.1 shows the assessment file for a student who has demonstrated the *radiant*-level of understanding in all but three learning objectives.

Reflections At the end of each assignment, we asked students to claim proficiency by submitting a "`META.md`" file containing claims of each rubric item on which they wanted to be assessed along with evidence of how their work supported their claims. We chose the file name "`META`" because it was the student providing a reflection – or metadata in a technical sense – apart from the work in the submission, and "`.md`" because we asked students to format it as a markdown file for readability in GitHub, which we used for homework management. After each homework submission, we provided feedback for each claim inline in the submitted `META.md` files.

The following is an example `META.md` file in which the student correctly provided evidence to document the claimed proficiency:

```
# Abstractions and Representations - Recognition
I considered and partially implemented a representation
of slideshow while writing my technical email.
In particular, I considered the possibility of using
type classes to ensure each instance of the Pict type
implements the various Pict primitives.
## FEEDBACK
This is an interesting point, because as other students
have pointed out, visual stuff often lends itself to
OOP style programming. [...]
```

Using (Functors, Applicatives, and) Monads

Due Wednesday, Sept 18th @ 9 am

From this assignment, students will show...

Functional Programming - Recall

through function definition and implementation.

CodeGen and Metaprogramming - Recall

through applying the lessons of code reuse from typeclasses to the three of the most common design patterns in haskell.

Metacognition, Reflection, and Personal Growth - Requisite

through answering the reflection questions in the META.md.

Note: For recall learning objectives, please tell me what skill you had to apply again and where it was originally applied.

A visual aid

I found this explination to be fun and helpful. Feel free to consult it if you get stuck.

Let's begin

1. Pull the latest Exemplars Github Repo
2. Follow the instructions in `src/Lib.hs` by completing all the tasks marked in the comments (e.g. `-- 1) ...`).
3. Answer the following reflection questions and add them to your `META.md`
   - From `hw01-1`, `hw02-1`, and this assignment, what coding topic or implementation do you feel you could most easily teach to another cs student at your level? Which do you feel would be most difficult to?
4. Finish the usual material for your META.md

Figure 6.2: An overview of a sample homework opening showing the rubric items suggested students earn on that assignment.

The instructor gave feedback to continue the conversation on the topic. In cases where proficiency was claimed but not awarded, instructors provided feedback explaining the rejection and how the student might earn the proficiency in future work.

Goal setting Figure 6.2 shows how each assignment started with a list of the rubric items we thought student work likely to exhibit. To show additional rubric items, students adjusted the assignment prompt for themselves, provided work that satisfied those adjustments, and stated claims of these additional rubric items in their `META.md` reflections. Students were not required to state in advance which items they were looking to achieve (or eschew), but evidence of their goal setting often appeared in their `META.md` reflections.

6.6 Results

6.6.1 Qualitative: Student `META.md`s over time

Early course `META.md`s At the beginning of the course, student `META.md`s primarily modeled the vocabulary given in the learning objectives of the assignment. For example, the second programming assignment, `hw01-1`, expected students to claim Functional Programming at the *Recall* level of understanding. The language provided for that level was "...by completing small functional programming examples to implement the familiar tools from scheme and ML." Most responses were explicit restatements of the prompt or variations without specific reference to assignment details, such as "implemented definitions for some basic functions." Exceptional student responses pointed specifically to the evidence. For example, "Used techniques from [PL prereq course] like pattern matching, type constructors, and understanding that the functions are curried, in order to implement these functions and algebraic data type in Haskell," and "I used functional programming techniques I learning in [PL prereq course] by writing the higher-order functions `filter`, `map`, and `foldr`, which take in another function as a parameter."

Mid course `META.md`s We made `META.md`s optional after `hw5` because some students reported difficulty reflecting on their work in the same turn-in as finishing the assignment. Students who continued to submit `META.md`s showed the anticipated benefits from claiming proficiency that explicitly referenced previous class sessions and peer comments. For example, a student claimed proficiency in their `META.md` accompanying a paper outline following an in-class, peer review activity: "Metacognition, Reflection, and Personal Growth - Reasoning: by incorporating the feedback [...], specifically, identifying what questions will be answered by each section." The student providing the peer review referenced a DSL evaluation criteria from a previous class activity, which was captured in student-authored class notes. The class activity had students analyze conference reviews of a DSL paper submission. We conclude this student was able to apply a toy version of reviewer feedback for a conference research paper, as interpreted by another student's peer review, to their own DSL writing.

Late course `META.md`s By the end of the course, students provided `META.md` claims of proficiency in three circumstances. The most common and consistent were students writing the `META.md` for its intrinsic value consistently throughout the course. The second most common was students claiming proficiency in levels of understanding they missed in earlier work. The least common was students claiming advanced proficiency.

6.6.2 Quantitative: Statistics on class work

Figure 6.3 shows that nearly all students submitted assignments and reflections early in the semester, many stopped submitting reflections mid-semester, and many stopped submitting assignments towards the end. All but one student submitted a final project and paper, but only 6 submitted a final video presentation summarizing their work for their peers. In the figure, we combined assignments `hw22` through `hw25` since they were a mix of rough draft and peer review assignments

related to the final paper that students adapted at their discretion.

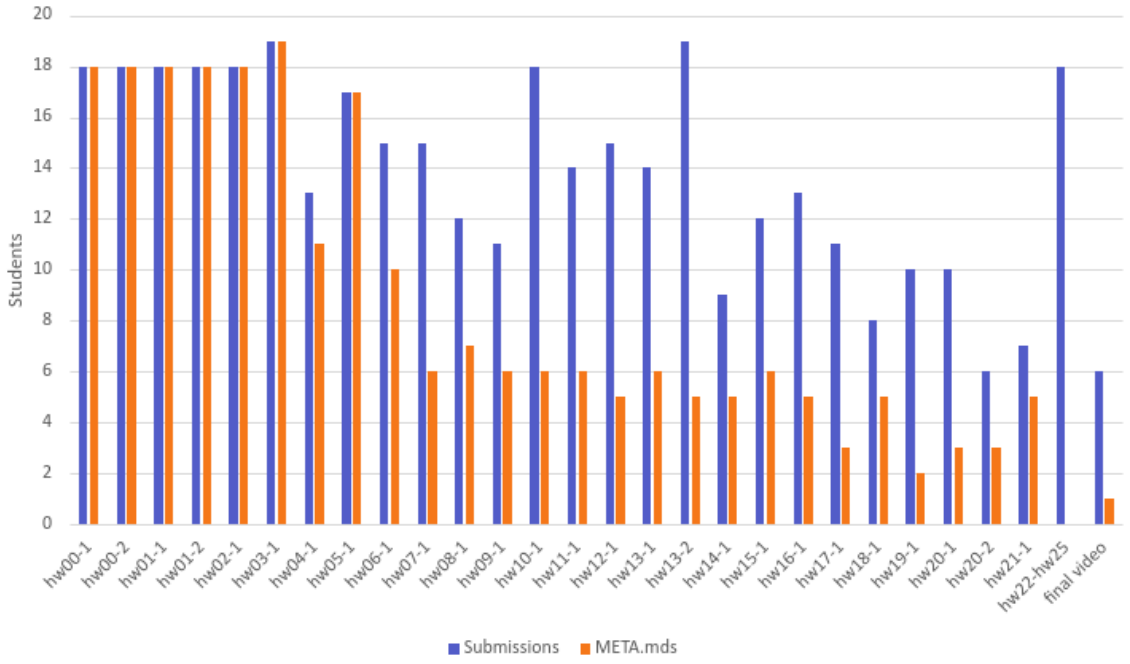


Figure 6.3: A comparison of how many students submitted project iterations and `META.mds` for each assignment (19 max).

Even without submitting every assignment, Table 6.1 shows most students (out of 19 possible) were awarded their proficiency claims in all learning objectives from the *requisite* through *reasoning* levels. The *radiant* learning objective (e.g. applying ideas from their project outside the course) primarily differentiated student performance, along with attendance and in-class participation. The outlier of Functional Programming at the *requisite* level having 18 students awarded while higher levels have 19 was caused by a student missing initial classes and never subsequently claiming proficiency (i.e., omitting any reflection on that learning from their `META.mds`).

Design and interpersonal skills were the most difficult for students to show proficiency in at the *radiant* level. Instead of showing proficiency in their projects, 8 out of the 19 students volunteered to lead the class in special topics of their own choosing towards the end of the course. Their preparation and reflections on the presentation provided evidence for claims of *radiant* proficiency in technical writing; documentation and usability; composing feedback and evaluation; and metacognition, reflection, and personal growth by reaching out to experts outside the course or discussing thought exercises of applying the concepts to their future work.

Table 6.2 shows how many rubric items were earned using activities or homework other than on a suggested one. These items may have been earned by showing learning in advanced of the suggested homework assignment, on a later homework assignment, or on an activity of the student's creation like a presentation on a special topic. We note that students exercised their agency more on rubric items with higher numbers. Students benefited mostly in the more abstract learning objectives, like feedback or personal growth, and in the middle levels of understanding, like recognition and reasoning. These metrics reflect our qualitative reasoning that student agency

	Req.	Recall	Recog.	Reason.	Radiant
SE&D	19	19	19	18	16
FP	18	19	19	17	11
A&R	19	19	19	18	17
TRC	19	19	19	19	15
TW&R	19	19	19	18	12
D&U	19	19	19	18	13
CG&M	19	19	17	17	13
LFED	19	19	19	17	12
CF&E	19	19	19	18	11
MR&PG	19	19	19	18	14

Table 6.1: Instances of proficiency exhibited in student work by the end of the course (19 max).

was seen as valuable towards the middle of the course especially as students overcame inertia to the teaching intervention. The successes in goal setting we saw during office hours, and the positive effects on student's proficiency self-belief, relates to the exercising of agency in the middle of the semester supported by these metrics.

	Req	Recall	Recog	Reason	Radiant
SE&D	1	3	7	6	3
FP	1	0	4	0	2
A&R	2	0	3	1	3
TRC	0	0	1	3	3
TW&R	5	4	11	1	0
D&U	1	3	5	14	0
CG&M	2	1	7	4	0
LFED	2	1	6	2	0
CF&E	2	5	5	12	1
MR&PG	13	7	14	2	6

Table 6.2: Rubric items earned on activities other than those suggested (19 max).

6.6.3 Student Feedback

At the end of the semester, 17 students submitted course evaluations. All quantitative measures for the course and staff were satisfactory or higher with the majority reporting the class was excellent (five on a five-point scale). Many students reported agency over their own learning, and their feedback revealed it took the combination of approaches to help everyone learn. This comment, for example, illustrates that students took away lessons of goal setting: "The stages of the [project] helped me plan better; if it had been all at once, I never would have gotten as far as I did."

Other comments show students' self-reported self-efficacy benefited from the course: "I not only implemented a DSL, I also had to think about "why" I used the tools that I used and how to evaluate my DSL." "I took away from this course a vocabulary and process of talking about

different design choices. There are questions I ask when considering a language that I wouldn't know to ask without this course." "The in class activities around the papers or other things [...] pushed me to think more critically [...] and to discuss my thoughts and exchange ideas with my classmates."

From the written responses, seven students talked specifically about applying methods of inquiry from this course to future problems. The remaining three talked about their active involvement in their learning to solve problems. In reporting what helped their learning, six students praised the course SRL structure in some way, four students credited the coding assignments and the project, while two students cited the reading. Three students mentioned feedback and interaction with the instructor. The most common complaint was the difficulty of using Haskell. Students requested more code examples, more practice opportunities, and less frequent homework due dates.

6.7 Discussion

6.7.1 Instructor experience

Grading effort When students chose to submit `META.md`s with their work, grading time was limited to spot checking the students' submission based on their guidance. Grading time was very short and uniform between students with variation between assignments stemming from the complexity of the evidence (which was proportional to the complexity of the homework task and under our control). When students did not submit a `META.md`, we looked instead for evidence to support the two to three learning objectives listed at the start of the homework. The small and frequent homework assignments assisted in the uniformity and kept search times short for those that did not submit `META.md`s.

Interactions and feedback with students The first three levels of understanding were perceived by students as "points everyone gets," but they enabled us to have rich feedback discussions without framing conversations in terms of points, grades, or redoing assignment activities. Notably, office hour conversations around missed rubric items not yet shown to be proficient were quickly refocused on content through discussing the given feedback and looking ahead at the schedule to the next opportunity to show that proficiency. Lack of familiarity meant students did not immediately understand the implications of the grading structure, with many asking questions about their grades with requests to "make up" assignments. Eventually, students understood that proficiency credit, once awarded, could not be taken away and future assignments were within their power to modify to show "missed" proficiency. Subsequent conversations with these students became increasingly goal-oriented.

Proficiency rubric trade-offs Considering Figures 6.1 and 6.3 together reveals the trade-offs in a course-wide proficiency rubric. Although students chose not to engage in every assignment, they showed proficiency in the learning objectives beyond our expectations. Also, no particular

assignment appeared obviously removable since each assignment helped some students show proficiency. For example, students who did not submit the video showed proficiency by going above and beyond in their final papers, project implementations, and special topics presentations. From the lack of final presentation videos, however, we collectively lost out on a sense of closure for the course. We conclude that our approach valued the active-learning process over its artifacts, but if instructors value any particular assignment intrinsically, then that assignment should have a completeness grade outside the proficiency rubric.

6.7.2 Student Experience

Student agency The teaching interventions intended framing agency where the initial state, goal state, and logical operations all contained substantial unknowns. The goal state of the seminar overall was a DSL of each student's own design that satisfied criteria the students collectively authored. The logical operators that students should take were provided in homework activities. However, the guarantee of the operators was that doing what was suggested would result in the skills needed for a minimally viable project. Otherwise, students had agency to follow, modify, or extend the operators provided in the homework to meet the specific needs of their project (that they must themselves determine). The initial state of the teaching interventions required students to choose a domain. As opposed to the "menu options" design choices where each choice is equivalent, but stylistically different, the choice of domain affects what particular DSL features, host language features, and independent knowledge students would need to learn about in order to meet the shared satisfying criteria of the end goal.

Students exhibited epistemic agency in all the dimensions, but most interesting in this study and not present in the others was agency for practice and for real-world application. By asking students to provide rationale for why they earned a particular proficiency, they felt encouraged to make more connections to prior practice in order to qualify their answers. That is, they would present extra information about choices they did not make and why, e.g. using or not using typeclasses for their implementation. This extra information often showed that students were making choices about which approaches might be better or worse based on their perception of existing practice. When instructors agreed, this would lead to affirming feedback. However, when instructors disagreed, this extra information provided a context the instructors would use to clarify perception about existing practice which helped the students make better decisions or better understand the goals of the course, e.g. compilation phases in the DSL vs making an interpreter in the style of the Intro to PL learning languages.

As the primary exhibition of epistemic agency, we saw that simply asking students to apply their learning to outside of the classroom led them to choose how to do so. While there was no expected way, the easiest way would be to apply the project to another course. Students determined different ways, however, like reaching out to library authors and leading discussions on special topics. In addition to the agency exhibited by these students, leaving the framing open allowed for other students to follow their peer's example to achieve the same proficiency, but with enough variation that there was no worries about over-fitting to each-others' work.

Goal setting Both complex and simple initial project designs required the same effort to satisfy the learning objectives of the course. Students chose tasks that benefited assessment and went above and beyond when they found intrinsic value. For students whose initial designs were overly ambitious, all but one was able to scope down non-learning-objective-earning features rather than switching to less ambitious projects, and the one student who did switch was ahead of the prescribed pace for learning objectives and wanted to explore a different domain. In contrast, two students who finished implementing simple initial designs well before the end of the course had not yet exhibited evidence for all proficiency items. They both found opportunities in the remaining assignments to improve their designs and implement additional features to earn the rest of the objectives.

Self-efficacy In their `META.md`s, students described their projects with more certain language as they referenced the results of in-class notes and reflections. When applying for proficiencies that were previously missed, for example, students manifested their confidence in the current attempt by explicitly comparing the two attempts. We did not require students to make these comparisons, but most who re-attempted work provided them.

In early in-class activities, students noticeably looked to the instructor for validation between statements of fact. In later classes, students instead referenced notes they collectively authored and each other rather than immediately asking for instructor intervention. In addition to simply gaining proficiency in speaking, we believe the reflection tasks – including collective note taking – fostered independent, active learning through improved self-efficacy.

6.7.3 Changes to future iterations

We saw that students who consistently reflected received better feedback and progress, despite most disliking reflection immediately upon completing the assignment. We do not suggest making reflections optional like we did after `hw05`. Instead, we recommend making reflections an independent task due after the corresponding homework.

Throughout the course, we found students using language such as "skipping", "make-up work", and "optional assignments" which our assessment structure did not support and which moved goal-setting conversations away from active participation and learning-oriented planning. In other words, these conversations focused on assignment grading and implied the assessment structure was tied to something other than the students' personal progress and learning. In a future iteration, we would work to better help students realize they are active participants and rewarded for exhibiting breadth and depth of learning rather than for the completeness of their implementation. Similarly, we anticipate adopting a different framing device, e.g. instead of "assignments" we would use "check-ins" to better support the metaphor. Otherwise we would not change the approach, and we would absolutely use it again for project-based seminar courses.

7

Conclusion

This dissertation presented how students may better retain Programming Language (PL) concepts beyond their initial learning environment through the correlated metric of self-efficacy. In studies with first-time programmers, we provided opportunities for students to exercise agency over the designs of projects. We found that students engaged with these design activities and made connections to the PL concepts from application domains. We observed students constructing a shared understanding between their peers, the activity materials, and the computing tools we provided, e.g. the DSLs we designed and the hardware students ran code on. We found that letting students make design decisions in the application domain provided positive vicarious experiences for students who had previously had low self-efficacy and participation during technical, skill building activities. Our study using the DSL BlockyTalky helped inform problems with the open-endedness and acceptance criteria offered by the design activity. By leaving acceptance criteria open-ended, students collectively constructed their own. When students provided their answers to the activity with instructors or the group, they presented their understanding with the acceptance criteria of syntactic correctness and some semantic correctness in the application domain. For example, grammatically correct sentences where the musical words made sense. When instructors attempted to help explore programming examples, they presented information with an implicit acceptance criteria of semantic expressiveness in the computational model. For example, something that could be turned into code. Mismatches between these understandings caused efforts for explanatory feedback from instructors to be interpreted as corrective feedback with the student's self-belief in their own accomplishments being challenged. When instructors (accidentally) limited student agency by modeling an example early in the activity for some participants, those students overfit their answers to the example. This overfitting obfuscates the shared knowledge creation of the group of students. As the primary takeaway, the study with BlockyTalky shows CS educators of first-time programmers that providing opportunities for student agency introduces opportunities for participation and collaborative knowledge creation typically unavailable in predetermined projects, but introducing agency requires careful scaffolding to build the correct acceptance criteria and explore the computational model without overfitting to examples.

Our study using the DSL Warble provided a possible solution to modeling a design activity with an explicit acceptance criteria while maintaining student agency. Students were still

asked to design projects that they were interested in so that they would engage and collaborate despite lack of technical proficiency demonstrated in the workshop so far. As an activity, a visual metaphor provided syntactic correctness acceptance criteria with the intention of being directly translated to a heavily restricted computational model. Students annotated illustrations of their project with this visual metaphor as a mechanism to explain how their projects would work, and instructors responded with feedback in terms of the metaphor so the acceptance criteria shared between the students and the instructors was the same. Students successfully translated their designs into working projects, but a combination of student contentment, time limitations, and visual clarity of the metaphor fostered simpler projects than what students were capable of. In the context of a workshop without curriculum constraints, the meta-acceptance criteria of exploring the computational model illustrates the challenge that increased student agency over project design imposes on far learning of PL concepts. Specifically, if we increase student agency to increase the intrinsic value in the projects, we should not be surprised when students create and act upon design goals that only serve project creation and not learning objectives. Restricting the application domain made progress towards fostering both student agency and interaction with the computational model, but future work could explore iterative design or other framing devices where students find value in expanding or elaborating on the projects and, hopefully, explore more of the computational model.

Our study with mid-level university students in an introductory programming languages course opened each of the weekly small group labs, called recitations, with critique activities on several existing, imperfect answers. This recitation activity intended to introduce student agency in an otherwise concept-driven curriculum design by giving students a small reflection opportunity to construct acceptance criteria with a group of their peers. Specifically, students were asked to evaluate existing answers based on if they agreed or disagreed with no intended perfectly correct discussion approach. We hoped that providing this opportunity would create positive vicarious experiences where students were not immediately judged to be correct or incorrect by their TAs or peers. From these experiences, we hoped that student participation and self-belief of how prepared they were to discuss the topics after the recitations would improve.

We found that this small intervention had no impact on participation as perceived by both TAs and students in comparison to simply opening lab with a question for corrective feedback. We found that, contrary to our hypothesis of topics introducing new syntax and semantics experiencing more benefit, instead the intervention increased discussion preparedness for topics using existing syntax and semantics on new computational models. We viewed increasing discussion preparedness relative to reading the book, writing code, and debugging code as beneficial since we believe that students start writing code before they are confident in the content of the topic and, when encountering difficulty, do not know if they need to ask a concept question or a proficiency question. Navigating epistemological messaging confusion consumes significant TA time during office hours and leads to the adverse quality of student and TA knowledge construction during office hours, which motivated this study. Continuing from this study, future work should explore teaching interventions on the PL topics that introduce new computational models in introductory PL courses. Student agency could benefit these topics by providing a more intuitive context for

the student to realize the topics in or relate the topics to things they already know similar to the algorithms study related work.

Our study on an upper-level programming language seminar had students learn how to read and discuss research papers on Domain-Specific Languages (DSL) and design, implement, and present prototype DSLs in domains of their choice as semester-long projects. We designed a two-dimensional, semester-long assessment rubric listing the content areas in which students needed to show proficiency and the levels of understanding at which they needed to show each. As a student-agency-focused grading strategy, we credited proficiency in a given rubric cell whenever a student demonstrated it. To support this assessment model, we structured a blended-learning environment; students completed reading, video lecture, and implementation as online assignments. Small group discussions and labs allowed students to practice their learning in class. At the end of class, students produced documentation in the form of wiki summary entries summarizing that class session's discussion. Students referenced this documentation in their reflection entries, and we instructors used the documentation to develop homework, such as what topics needed more examples and what the DSL design evaluation criteria should be. Along with homework assignments, students produced reflections, which they used as the assessment instrument and we used to determine the needs of future class activities. This feedback loop formed our integration between the online and in-class components.

Students successfully implemented DSLs of their own design in application domains of their own choosing. Students achieved far transfer oriented learning objectives at increasing levels of understanding. Students were successfully able to tailor the homework assignments to the objectives they had yet to show, and omitted homework assignments they believed would not help them learn new concepts. Students who found intrinsic value in assignments, even when not demonstrating new understanding in learning objectives for assessment, participated primarily in service to their peers in the course. Students achieved concrete instances of open ended goals often beyond instructor expectations as our rubric treated students as co-authorities in their learning. Adding this open-endedness had little impact overall on grading or teaching effort compared to anticipated seminar style facilitation with a semester-long project. Time spent in advance on goal setting and building a shared learning context eliminated time typically spent on make-up work or the redundancy of "catching up", allowing for students to perceive their work by always looking forward. Due to the novelty of this approach, students presented initial inertia to the assessment model. For example, requests for make up assignments and resubmission of work were made initially when, by design, learning objectives yet to be demonstrated could be set as goals in the upcoming assignments. These requests highlight how the extrinsic value of assignments from other assessment models conflicts with the intended intrinsic value in service of far learning. For example, assigned reading for credit on comprehension questions vs for preparation of the following in-class discussion and future work's acceptance criteria.

Students made connections to the application domain as it exists in the wild by authoring acceptance criteria for the DSLs presented in the research papers and applying them to their own DSLs. They experienced general programming practices such as finding, utilizing, and describing publicly available domain libraries and interacting with open-source project maintainers, which

supports our goals that the learning will be re-applied outside the context of the class' learning environment. We hope that when students apply these practices again, they rely on their projects from this seminar as concrete examples and relate the PL concepts they learned from authoring DSLs for this project to their professional context. However, we did see problems arise from epistemology of science-like sources when connecting to learning outside the classroom as students initially authored unrealistic acceptance criteria for their DSL designs in comparison with the DSLs presented in the wild for their chosen application domains and in research papers. From these results, we can improve the learning environments for earlier experience levels by scaffolding the most successful student agency activities like reaching out to code library authors, connecting the work to other CS courses, and asking students to explore application domains in addition to activities specifically crafted to illustrate PL concepts.

Bibliography

[ARS96] J. R. Anderson, L. M. Reder, and H. A. Simon, "Situated learning and education", *Educational Researcher*, vol. 25, no. 4, pp. 5–11, 1996.

[And19] C. Andrade, "The p value and statistical significance: Misunderstandings, explanations, challenges, and alternatives", eng, *Indian journal of psychological medicine*, vol. 41, no. 3, pp. 210–215, 2019, IJPsyM-41-210[PII], ISSN: 0253-7176. DOI: 10.4103/IJPSYM.IJPSYM_193_19. [Online]. Available: https://doi.org/10.4103/IJPSYM.IJPSYM_193_19.

[BSD14] S. Baltes, P. Schmitz, and S. Diehl, "Linking sketches and diagrams to source code artifacts", in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, Hong Kong, China: Association for Computing Machinery, 2014, pp. 743–746, ISBN: 9781450330565. DOI: 10.1145/2635868.2661672. [Online]. Available: https://doi.org/10.1145/2635868.2661672.

[Ban97] A. Bandura, *Self-efficacy: The exercise of control*. Ser. Self-efficacy: The exercise of control. New York, NY, US: W H Freeman/Times Books/ Henry Holt & Co, 1997, pp. ix, 604–ix, 604, ISBN: 0-7167-2626-2 (Hardcover); 0-7167-2850-8 (Paperback).

[BS81] A. Bandura and D. H. Schunk, "Cultivating competence, self-efficacy, and intrinsic interest through proximal self-motivation", *Journal of personality and social psychology*, vol. 41, no. 3, pp. 586–598, Sep. 1981.

[BC09] E. Baran and A.-P. Correia, "Student-led facilitation strategies in online discussions", *Distance Education*, vol. 30, no. 3, pp. 339–361, 2009. DOI: 10.1080/01587910903236510. [Online]. Available: https://doi.org/10.1080/01587910903236510.

[BC02] S. M. Barnett and S. J. Ceci, "When and where do we apply what we learn?: A taxonomy for far transfer", English, *Psychological bulletin*, vol. 128, no. 4, pp. 612–637, Jul. 2002.

[BLM+18] M. Behroozi, A. Lui, I. Moore, D. Ford, and C. Parnin, "Dazed: Measuring the cognitive load of solving technical interview problems at the whiteboard", in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 93–96, ISBN: 9781450356626. DOI: 10.1145/3183399.3183415. [Online]. Available: https://doi.org/10.1145/3183399.3183415.

[BNA15] M. L. Bernacki, T. Nokes-malach, and V. Aleven, "Examining self-efficacy during learning: Variability and relations to behavior, performance, and learning", English, *Metacognition and Learning*, vol. 10, no. 1, pp. 99–117, Apr. 2015.

[BF06] A. Blandford and D. Furniss, "Dicot: A methodology for applying distributed cognition to the design of teamworking systems", in *Interactive Systems. Design, Specification, and Verification*, S. W. Gilroy and M. D. Harrison, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 26–38, ISBN: 978-3-540-34146-8.

[Bra03] B. Bracken, "Progressing from student to professional: The importance and challenges of teaching software engineering", *J. Comput. Sci. Coll.*, vol. 19, no. 2, pp. 358–368, Dec. 2003, ISSN: 1937-4771.

[BBC99] J. D. Bransford, A. L. Brown, and R. R. Cocking, *How people learn: Brain, mind, experience, and school.* National Academy Press, 1999.

[Col98] M. Cole, *Cultural psychology: A once and future discipline*. Harvard University Press, 1998.

[CBN89] A. Collins, J. Brown, and S. Newinan, "Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics", in *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, Lawrence Erlbaum Associates, Inc., 1989, pp. 453–494.

[Daw17] P. Dawson, "Assessment rubrics: Towards clearer and more replicable design, research and practice", *Assessment & Evaluation in Higher Education*, vol. 42, no. 3, pp. 347–360, 2017. DOI: 10.1080/02602938.2015.1111294. eprint: https://doi.org/10.1080/02602938.2015.1111294. [Online]. Available: https://doi.org/10.1080/02602938.2015.1111294.

[DCBB15] T. Degueule, B. Combemale, A. Blouin, and O. Barais, "Reusing legacy dsls with melange", in *Proceedings of the Workshop on Domain-Specific Modeling*, ser. DSM 2015, Pittsburgh, PA, USA: Association for Computing Machinery, 2015, pp. 45–46, ISBN: 9781450339032. DOI: 10.1145/2846696.2846697. [Online]. Available: https://doi.org/10.1145/2846696.2846697.

[DSA+15] E. Deitrick, R. B. Shapiro, M. P. Ahrens, R. Fiebrink, P. D. Lehrman, and S. Farooq, "Using distributed cognition theory to analyze collaborative computer science learning", in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ser. ICER '15, Omaha, Nebraska, USA: Association for Computing Machinery, 2015, pp. 51–60, ISBN: 9781450336307. DOI: 10.1145/2787622.2787715. [Online]. Available: https://doi.org/10.1145/2787622.2787715.

[DKI08] N. Diamond, S. K. Koernig, and Z. Iqbal, "Uniting active and deep learning to teach problem-solving skills: Strategic tools and the learning spiral", English, *Journal of Marketing Education*, vol. 30, no. 2, pp. 116–129, Aug. 2008.

[DiS14] B. DiSalvo, "Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science", *IEEE Computer Graphics and Applications*, vol. 34, no. 6, pp. 12–15, 2014. DOI: 10.1109/MCG.2014.112.

[Eas69] C. M. Eastman, "Cognitive processes and ill-defined problems: A case study from design", in *Proceedings of the International Joint Conference on Artificial Intelligence: IJCAI*, Citeseer, vol. 69, 1969, pp. 669–690.

[EK17] E. Enström and V. Kann, "Iteratively intervening with the "most difficult" topics of an algorithms and complexity course", *ACM Trans. Comput. Educ.*, vol. 17, no. 1, Jan. 2017. DOI: 10.1145/3018109. [Online]. Available: https://doi-org.ezproxy.library.tufts.edu/10.1145/3018109.

[FVF14] K. Falkner, R. Vivian, and N. J. Falkner, "Identifying computer science self-regulated learning strategies", in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ser. ITiCSE '14, Uppsala, Sweden: Association for Computing Machinery, 2014, pp. 291–296, ISBN: 9781450328333. DOI: 10.1145/2591708.2591715.

[FW11] K. Fisher and D. Walker, "The pads project: An overview", in *Proceedings of the 14th International Conference on Database Theory*, ser. ICDT '11, Uppsala, Sweden: Association for Computing Machinery, 2011, pp. 11–17, ISBN: 9781450305297. DOI: 10.1145/1938551.1938556. [Online]. Available: https://doi.org/10.1145/1938551.1938556.

[FMM+14] J. Freeman, B. Magerko, T. McKlin, M. Reilly, J. Permar, C. Summers, and E. Fruchter, "Engaging underrepresented groups in high school introductory computing through computational remixing with earsketch", in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14, Atlanta, Georgia, USA: Association for Computing Machinery, 2014, pp. 85–90, ISBN: 9781450326056. DOI: 10.1145/2538862.2538906. [Online]. Available: https://doi.org/10.1145/2538862.2538906.

[GB04] K. Garvin-Doxas and L. J. Barker, "Communication in computer science classrooms: Understanding defensive climates as a means of creating supportive behaviors", *J. Educ. Resour. Comput.*, vol. 4, no. 1, 2–es, Mar. 2004, ISSN: 1531-4278. DOI: 10.1145/1060071.1060073. [Online]. Available: https://doi.org/10.1145/1060071.1060073.

[GWMB04] A. Gavrin, J. X. Watt, K. Marrs, and R. E. Blake Jr, "Just-in-time teaching (jitt): Using the web to enhance classroom learning", *Computers in education journal*, vol. 14, no. 2, pp. 51–60, 2004.

[Guz21] M. Guzdial, "Reaching everyone by integrating computing everywhere", in *Proceedings of the 10th Computer Science Education Research Conference*, ser. CSERC '21, Virtual Event, Netherlands: Association for Computing Machinery, 2021, pp. 3–4,

ISBN: 9781450385763. DOI: 10.1145/3507923.3507925. [Online]. Available: https://doi.org/10.1145/3507923.3507925.

[HIP+18] A. Hellas, P. Ihantola, A. Petersen, V. V. Ajanovski, M. Gutica, T. Hynninen, A. Knutas, J. Leinonen, C. Messom, and S. N. Liao, "Predicting academic performance: A systematic literature review", in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE 2018 Companion, Larnaca, Cyprus: Association for Computing Machinery, 2018, pp. 175–199, ISBN: 9781450362238. DOI: 10.1145/3293881.3295783. [Online]. Available: https://doi-org.ezproxy.library.tufts.edu/10.1145/3293881.3295783.

[HQ03] C. L. Holladay and M. A. Quiñones, "Practice variability and transfer of training: The role of self-efficacy generality", eng, *Journal of applied psychology*, vol. 88, no. 6, pp. 1094–1103, 2003, ISSN: 0021-9010.

[HHK00] J. Hollan, E. Hutchins, and D. Kirsh, "Distributed cognition: Toward a new foundation for human-computer interaction research", *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 2, pp. 174–196, 2000.

[Hra19] S. Hrastinski, "What do we mean by blended learning?", *TechTrends*, vol. 63, pp. 564–569, 2019. [Online]. Available: https://doi.org/10.1007/s11528-019-00375-5.

[Hud96] P. Hudak, "Building domain-specific embedded languages", *ACM Comput. Surv.*, vol. 28, no. 4es, 196–es, Dec. 1996, ISSN: 0360-0300. DOI: 10.1145/242224.242477. [Online]. Available: https://doi.org/10.1145/242224.242477.

[Hut95] E. Hutchins, *Cognition in the Wild*. MIT press, 1995.

[Jon00] D. H. Jonassen, "Toward a design theory of problem solving", *Educational technology research and development*, vol. 48, no. 4, pp. 63–85, 2000.

[JOP+–] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python*, 2001–. [Online]. Available: http://www.scipy.org/.

[JRR+09] C. Justice, J. Rice, D. Roy, B. Hudspith, and H. Jenkins, "Inquiry-based learning in higher education: Administrators' perspectives on integrating inquiry pedagogy into the curriculum", *Higher Education*, vol. 58, no. 6, p. 841, Mar. 2009, ISSN: 1573-174X. DOI: 10.1007/s10734-009-9228-7. [Online]. Available: https://doi.org/10.1007/s10734-009-9228-7.

[KBC15] D. Kafura, A. C. Bart, and B. Chowdhury, "Design and preliminary results from a computational thinking course", in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '15, Vilnius, Lithuania: Association for Computing Machinery, 2015, pp. 63–68, ISBN: 9781450334402. DOI: 10.1145/2729094.2742593. [Online]. Available: https://doi.org/10.1145/2729094.2742593.

[KLT22] D. Kirk, A. Luxton-Reilly, and E. Tempero, "Risks in student projects", in *Australasian Computing Education Conference*, ser. ACE '22, Virtual Event, Australia: Association for Computing Machinery, 2022, pp. 143–152, ISBN: 9781450396431. DOI: 10.1145/3511861.3511877. [Online]. Available: https://doi.org/10.1145/3511861.3511877.

[Kit83] K. S. Kitchner, "Cognition, metacognition, and epistemic cognition", *Human development*, vol. 26, no. 4, pp. 222–232, 1983.

[Kme15] E. Kmett, *Kmett/propagators*, https://https://github.com/ekmett/propagators, Accessed: 2022-07-04, 2015.

[Lav88] J. Lave, *Cognition in practice: Mind, mathematics and culture in everyday life*. Cambridge University Press, 1988.

[LGP13] C. B. Lee, S. Garcia, and L. Porter, "Can peer instruction be effective in upper-division computer science courses?", *ACM Trans. Comput. Educ.*, vol. 13, no. 3, Aug. 2013. DOI: 10.1145/2499947.2499949. [Online]. Available: https://doi.org/10.1145/2499947.2499949.

[Lew10] C. M. Lewis, "How programming environment shapes perception, learning and goals: Logo vs. scratch", in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10, Milwaukee, Wisconsin, USA: Association for Computing Machinery, 2010, pp. 346–350, ISBN: 9781450300063. DOI: 10.1145/1734263.1734383. [Online]. Available: https://doi.org/10.1145/1734263.1734383.

[LSA+18] A. Luxton-Reilly, Simon, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, "Introductory programming: A systematic literature review", in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE 2018 Companion, Larnaca, Cyprus: Association for Computing Machinery, 2018, pp. 55–106, ISBN: 9781450362238. DOI: 10.1145/3293881.3295779. [Online]. Available: https://doi.org/10.1145/3293881.3295779.

[MW47] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other", *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947, ISSN: 00034851. [Online]. Available: http://www.jstor.org/stable/2236101.

[Mar06] J. S. Marzano Robert J.; Kendall, *The New Taxonomy of Educational Objectives*. Thousand Oaks, California: Corwin Press, 2006.

[ME89] J. A. Meacham and N. C. Emont, "The interpersonal basis of everyday problem solving.", 1989.

[MMR+18] E. Miller, E. Manz, R. Russ, D. Stroupe, and L. Berland, "Addressing the epistemic elephant in the room: Epistemic agency and the next generation science standards", *Journal of Research in Science Teaching*, vol. 55, no. 7, pp. 1053–1075, 2018.

[MCL+21] A. Milliken, V. Cateté, A. Limke, I. Gransbury, H. Chipman, Y. Dong, and T. Barnes, "Exploring and influencing teacher grading for block-based programs through rubrics and the gradesnap tool", in *Proceedings of the 17th ACM Conference on International Computing Education Research*, ser. ICER 2021, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 101–114, ISBN: 9781450383264. DOI: 10.1145/3446871.3469762. [Online]. Available: https://doi.org/10.1145/3446871.3469762.

[Min92] S. Minör, "Interacting with structure-oriented editors", *Int. J. Man-Mach. Stud.*, vol. 37, no. 4, pp. 399–418, Oct. 1992, ISSN: 0020-7373. DOI: 10.1016/0020-7373(92)90002-3. [Online]. Available: https://doi.org/10.1016/0020-7373(92)90002-3.

[Mor04] R. Moreno, "Decreasing cognitive load for novice students: Effects of explanatory versus corrective feedback in discovery-based multimedia.", *Instructional Science*, vol. 32, 2004. DOI: 10.1023/B:TRUC.0000021811.66966.1d. [Online]. Available: https://doi.org/10.1023/B:TRUC.0000021811.66966.1d.

[Pac01] M. Packer, "The problem of transfer, and the sociocultural critique of schooling", *The Journal of the Learning Sciences*, vol. 10, no. 4, pp. 493–514, 2001.

[Pat01] R. Paterson, "A new notation for arrows", in *International Conference on Functional Programming*, Firenze, Italy: ACM Press, Sep. 2001, pp. 229–240. [Online]. Available: http://www.soi.city.ac.uk/~ross/papers/notation.html.

[PWW04] S. R. Porter, M. E. Whitcomb, and W. H. Weitzer, "Multiple surveys of students and survey fatigue", *New Directions for Institutional Research*, vol. 2004, no. 121, pp. 63–73, 2004. DOI: https://doi.org/10.1002/ir.101. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ir.101. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ir.101.

[Pri04] M. Prince, "Does active learning work? a review of the research", *Journal of Engineering Education*, vol. 93, pp. 223–231, Jul. 2004. DOI: 10.1002/j.2168-9830.2004.tb00809.x.

[Puc+96] M. Puckette *et al.*, "Pure data: Another integrated computer music environment", *Proceedings of the second intercollege computer music concerts*, pp. 37–41, 1996.

[Rad09] A. Radul, "Propagation networks: A flexible and expressive substrate for computation", 2009. [Online]. Available: http://groups.csail.mit.edu/genesis/papers/radul%5C%202009.pdf.

[RW98] V. Ramalingam and S. Wiedenbeck, "Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy", *Journal of Educational Computing Research*, vol. 19, no. 4, pp. 367–381, 1998. DOI: 10.2190/C670-Y3C8-LTJ1-CT3P. [Online]. Available: https://doi.org/10.2190/C670-Y3C8-LTJ1-CT3P.

[RKF19] Y. Ren, S. Krishnamurthi, and K. Fisler, "What help do students seek in ta office hours?", in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ser. ICER '19, Toronto ON, Canada: Association for Computing Machinery, 2019, pp. 41–49, ISBN: 9781450361859. DOI: 10.1145/3291279.3339418. [Online]. Available: https://doi-org.ezproxy.library.tufts.edu/10.1145/3291279.3339418.

[RMM+09] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, *et al.*, "Scratch: Programming for all", *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[RO+90] M. Resnick, S. Ocko, *et al.*, *LEGO/logo–learning through and about design*. Epistemology and Learning Group, MIT Media Laboratory Cambridge, 1990.

[Ria20] V. V. Riabov, "What makes students' capstone projects successful?", *J. Comput. Sci. Coll.*, vol. 35, no. 8, pp. 280–282, Apr. 2020, ISSN: 1937-4771.

[Rin08] M. Rinard, "Using programming language concepts to teach general thinking skills", *SIGPLAN Not.*, vol. 43, no. 11, pp. 111–118, Nov. 2008, ISSN: 0362-1340. DOI: 10.1145/1480828.1480853. [Online]. Available: https://doi.org/10.1145/1480828.1480853.

[Rog03] B. Rogoff, *The cultural nature of human development*. Oxford University Press, 2003.

[Rus14] R. S. Russ, "Epistemology of science vs. epistemology for science.", *Science Education*, 2014.

[Rus18] ——, "Characterizing teacher attention to student thinking: A role for epistemological messages", *Journal of Research in Science Teaching*, vol. 55, no. 1, pp. 94–120, 2018. DOI: https://doi.org/10.1002/tea.21414. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/tea.21414.

[Sax88] G. B. Saxe, "Candy selling and math learning", *Educational Researcher*, vol. 17, no. 6, pp. 14–21, 1988. DOI: 10.3102/0013189X017006014. eprint: https://doi.org/10.3102/0013189X017006014. [Online]. Available: https://doi.org/10.3102/0013189X017006014.

[SA16] R. B. Shapiro and M. Ahrens, "Beyond blocks: Syntax and semantics", *Commun. ACM*, vol. 59, no. 5, pp. 39–41, Apr. 2016, ISSN: 0001-0782. DOI: 10.1145/2903751. [Online]. Available: https://doi.org/10.1145/2903751.

[SKAF16] R. B. Shapiro, A. Kelly, M. Ahrens, and R. Fiebrink, "Blockytalky: A physical and distributed computer music toolkit for kids", in *International Conference on New Interfaces for Musical Expression*, We thank the National Science Foundation (CNS-1418463), the NCWIT Acadamic Alliance Seed Fund, and LEGO Education for funding this work., NIME, Jul. 2016. [Online]. Available: https://research.gold.ac.uk/id/eprint/18635/.

[SKA+17] R. B. Shapiro, A. Kelly, M. Ahrens, B. Johnson, H. Politi, and R. Fiebrink, "Tangible Distributed Computer Music for Youth", *Computer Music Journal*, vol. 41, no. 2, pp. 52–68, Jun. 2017, ISSN: 0148-9267. DOI: 10.1162/COMJ_a_00420. eprint: https://direct.mit.edu/comj/article-pdf/41/2/52/1856403/comj_a_00420.pdf. [Online]. Available: https://doi.org/10.1162/COMJ%5C_a%5C_00420.

[SJ02] T. Sheard and S. P. Jones, "Template meta-programming for haskell", in *Proceedings of the 2002 ACM SIGPLAN Workshop on Haskell*, ser. Haskell '02, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2002, pp. 1–16, ISBN: 1581136056. DOI: 10.1145/581690.581691. [Online]. Available: https://doi.org/10.1145/581690.581691.

[SMC+09] S. Sheppard, K. Macatangay, A. Colby, W. M. Sullivan, and L. S. Shulman, *Educating engineers: Designing for the future of the field*. Jossey-Bass San Francisco, CA, 2009, vol. 9.

[SWA+09] M. K. Smith, W. B. Wood, W. K. Adams, C. Wieman, J. K. Knight, N. Guild, and T. T. Su, "Why peer discussion improves student performance on in-class concept questions", *Science*, vol. 323, no. 5910, pp. 122–124, 2009. DOI: 10.1126/science.1165919.

[SDR94] J. P. Smith III, A. A. Disessa, and J. Roschelle, "Misconceptions reconceived: A constructivist analysis of knowledge in transition", *The Journal of the Learning Sciences*, vol. 3, no. 2, pp. 115–163, 1994.

[SS13] A. Stefik and S. Siebert, "An empirical investigation into programming language syntax", *ACM Trans. Comput. Educ.*, vol. 13, no. 4, Nov. 2013. DOI: 10.1145/2534973. [Online]. Available: https://doi.org/10.1145/2534973.

[SP20] V. Svihla and T. B. Peele-Eady, "Framing agency as a lens into constructionist learning", *Proceedings of Constructionism 2020*, 2020.

[TV18] L. Toma and J. Vahrenhold, "Self-efficacy, cognitive load, and emotional reactions in collaborative algorithms labs - a case study", in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ser. ICER '18, Espoo, Finland: Association for Computing Machinery, 2018, pp. 1–10, ISBN: 9781450356282. DOI: 10.1145/3230977.3230980. [Online]. Available: https://doi-org.ezproxy.library.tufts.edu/10.1145/3230977.3230980.

[TYC12] F. S. Tsai, C. Yuen, and N.-M. Cheung, "Interactive learning in pre-university mathematics", in *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) 2012*, 2012, T2D-8-T2D–13. DOI: 10.1109/TALE.2012.6360393.

[Val18] R. Vallat, "Pingouin: Statistics in python", *The Journal of Open Source Software*, vol. 3, no. 31, p. 1026, Nov. 2018.

[vDKV00] A. van Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography", *SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, Jun. 2000, ISSN: 0362-1340. DOI: 10.1145/352029.352035. [Online]. Available: https://doi-org.ezproxy.library.tufts.edu/10.1145/352029.352035.

[VD00] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong.", *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000. DOI: 10.2307/1165329. [Online]. Available: https://doi.org/10.2307/1165329.

[Vau21] M. Vaughn, *Student Agency in the Classroom : Honoring Student Voice in the Curriculum*, eng. New York, NY: Teachers College Press, 2021, ISBN: 0807779741.

[Vel21] J. Á. Velázquez-Iturbide, "An analysis of the formal properties of bloom's taxonomy and its implications for computing education", in *21st Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '21, Joensuu, Finland: Association for Computing Machinery, 2021, ISBN: 9781450384889. DOI: 10.1145/3488042.3488069.

[VB14] S. Vossoughi and B. Bevan, "Making and tinkering: A review of the literature", *National Research Council Committee on Out of School Time STEM*, vol. 67, pp. 1–55, 2014.

[Vyg80] L. S. Vygotsky, *Mind in society: The development of higher psychological processes*. Harvard University Press, 1980.

[Wei15] D. Weintrop, "Minding the gap between blocks-based and text-based programming (abstract only)", in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15, Kansas City, Missouri, USA: Association for Computing Machinery, 2015, p. 720, ISBN: 9781450329668. DOI: 10.1145/2676723.2693622. [Online]. Available: https://doi.org/10.1145/2676723.2693622.

[WW15] D. Weintrop and U. Wilensky, "To block or not to block, that is the question: Students' perceptions of blocks-based programming", in *Proceedings of the 14th International Conference on Interaction Design and Children*, ser. IDC '15, Boston, Massachusetts: Association for Computing Machinery, 2015, pp. 199–208, ISBN: 9781450335904. DOI: 10.1145/2771839.2771860. [Online]. Available: https://doi.org/10.1145/2771839.2771860.

[WFC99] T. L. Wenzlaff, J. J. Fager, and M. J. Coleman, "What is a rubric? do practitioners and the literature agree?", *Contemporary education*, vol. 70, no. 4, p. 41, 1999.

[Woo83] P. K. Wood, "Inquiring systems and problem structure: Implications for cognitive development", *Human development*, vol. 26, no. 5, pp. 249–265, 1983.

[Zim08] B. J. Zimmerman, "Investigating self-regulation and motivation: Historical background, methodological developments, and future prospects", English, *American Educational Research Journal*, vol. 45, no. 1, pp. 166–183, Mar. 2008.

[ZP86] B. J. Zimmerman and M. M. Pons, “Development of a structured interview for assessing student use of self-regulated learning strategies”, *American Educational Research Journal*, vol. 23, no. 4, pp. 614–628, 1986. DOI: 10.3102/00028312023004614. [Online]. Available: https://doi.org/10.3102/00028312023004614.